TM-71-1025-1

TECHNICAL MEMORANDUM

THE AUTOMATED TASK SCHEDULER SYSTEMS MANUAL

Bellcomm

BELLCOMM, INC.
955 L'ENFANT PLAZA NORTH, S.W. WASHINGTON, D.C. 20024

COVER SHEET FOR TECHNICAL MEMORANDUM

TITLE- The Automated Task Scheduler Systems Manual

TM- 71-1025-1

DATE- February 12, 1971

FILING CASE NO(S)- 610

AUTHOR(S)- A. B. Baker

FILING SUBJECT(S)- Flight Planning, Mission Timelining,
(ASSIGNED BY AUTHOR(S)- Resource Allocation, Scheduling

ABSTRACT

This report presents the system level documentation on the Automated Task Scheduler (ATS) System, a set of computer programs designed to produce and display timelines of in-flight activities for manned space missions. The programs are intended to be used for investigations into the process of flight scheduling.

The system consists of three programs: a Schedule Generator, a Data Processor, and a Data Bank Generator. The Schedule Generator (46,200 words) is the major program in the system. It uses a "window-filling" technique to generate time histories of allocations for each of several designated resources (e.g., crewmen, electrical power, pieces of equipment, etc.). The program may be used to generate a completely new schedule or to complete a partial schedule generated by a previous run. The latter feature can be used to investigate the effects of variations in a basically desirable schedule. The Data Processor (35,100 words) is used to produce graphical displays of the timeline data produced by the Schedule Generator while the Data Bank Generator (22,800 words) is used to create and maintain a Task Data Bank, a data library containing descriptions of all tasks that may be considered for scheduling on a given mission.

The system has been implemented on Bellcomm's UNIVAC-1108 computer. The programs are written primarily in FORTRAN V, SAC-1 (a FORTRAN-imbedded list-processing language), and 1108 Assembly Language. List-processing and dynamic array storage were used to increase the efficiency of computer storage utilization.

TABLE OF CONTENTS

BELLCOMM, INC.
955 L'ENFANT PLAZA NORTH, S.W. WASHINGTON, D.C. 20024

SUBJECT: The Automated Task Scheduler Systems Manual - Case 610

DATE: February 12, 1971

FROM: A. B. Baker

TM-71-1025-1

TECHNICAL MEMORANDUM

1.0 Introduction

The Automated Task Scheduler (ATS) System is a group of computer programs designed to produce and display mission timelines (schedules) for manned space missions at the level of detail normally found in a flight plan. The system is intended to be used primarily for investigations of the flight scheduling process.

The incentive to develop the ATS stems from the increase in the duration of future manned space missions compared to those in previous missions. Flight scheduling for the relatively short duration missions in the Mercury, Gemini, and Apollo Programs was performed manually. However, missions in the Skylab Program are to last for one to two months and missions of even longer duration are planned for the post-Skylab period. The increased duration of these missions will significantly increase the complexity of scheduling. This increase provides the motivation for attempting to automate as much of the scheduling process as possible in order to: (1) reduce the burden of tedious manual scheduling, and (2) decrease the time required to construct detailed timelines.

At the beginning of the present effort, a review of the state-of-the-art found that a number of automated schedulers had already been built (Reference 1). The review provided several concepts that have been used in the development of the ATS. These include:

1. That the model be organized into three distinct functional areas -- Input and Data Preparation, Scheduler, and Processor.

2. That the model be sufficiently modular so that each function and major subfunction is as isolated as possible from the rest of the model. This structure facilitates evaluation of different computational techniques in each area.

3. That data libraries or "data banks" be established which would contain the large amounts of input data required by a scheduler. The banks, stored on magnetic tape or FASTRAND file, would simplify the input card decks for each computer run.

4. That spacecraft ephemeris data be generated independently of the scheduler.

5. That provision be made for analysis of timelines generated by the scheduler.

2.0 The ATS Scheduling Algorithm

The primary objective of an automated scheduler is to assemble a given set of tasks into a self-consistent timeline within the structure defined by mission constraints, subsystem capabilities, and inter-task constraints. In the ATS a task is described, in part, by a set of resource* allocation requests. Each request or requirement specifies that a particular resource be allocated to the task during each performance of the task. Therefore, the ATS does not generate a single mission timeline but rather a set of time-lines, each timeline describing the time history of the allocations of one resource over the duration of the mission.

The timelines are produced by repeated cyclings through a basic three-step sequence:

Step 1: Select a task from those tasks not yet considered.

Step 2: Identify opportunities where the candidate task can be scheduled.

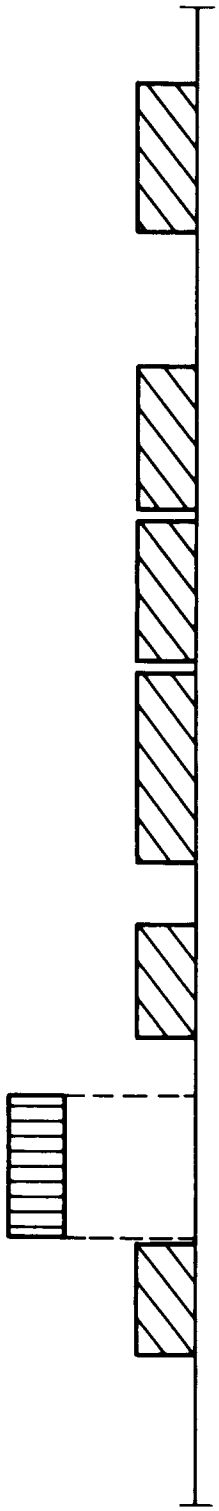
Step 3: Select one or more opportunities and commit the task to those places in the appropriate resource time-lines.

After the candidate task is selected, the entire mission duration is searched for acceptable scheduling opportunities. Task performances may then be scheduled at any of these opportunities. The method of scheduling tasks at acceptable opportunities anywhere over the mission duration is known as the "window-filling" scheduling technique. It is illustrated in Figure 2.1.

Note that the sequence is repeated once for each task. At that time a decision is made whether or not to schedule the candidate task and if so, which of the scheduling opportunities to utilize. Once made, the decisions are irrevocable for the remainder of the scheduling process (one computer 'run').

---

*Examples of resources include a crewman, a piece of equipment, electrical power, etc.



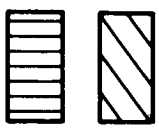


FIGURE 2.1 – THE WINDOW-FILLING SCHEDULING TECHNIQUE.

2.1 Selection of a Candidate Task

A task's position in the sequence of candidate selection will have a marked effect on the number of scheduling opportunities available for that task. Hence, some method should be available to select the candidate tasks in the most advantageous order, i.e., in a sequence that maximizes the number of tasks that are scheduled.

One particularly useful method of ranking consists of assigning a number to each task. Tasks are then ranked according to the relative magnitudes of these assigned numbers (or priorities) and considered for scheduling in the resulting order. There are two basic types of priorities: static and dynamic. Static priorities are assigned to each task by the user before the scheduling process begins. The numerical values represent the user's estimate of the relative difficulty of scheduling and remain constant throughout the scheduling process.

As noted above, the number of scheduling opportunities available for any task depends upon the commitments already made at the time the task is being considered. As the scheduling process continues, and more commitments are made, the number of scheduling opportunities for each task will decrease, and hence the difficulty with which the tasks can be scheduled will increase. Rather than remain constant, the priorities for the remaining tasks might be revised each time a task is scheduled to reflect changes in scheduling difficulty. Candidate tasks would then be selected for scheduling in descending order of the latest assigned priorities. Priorities that are revised during the scheduling process are called "dynamic" priorities.

An optimum ranking system would combine the features of the static and dynamic systems. The difficulty in implementing such a system stems from the inability to define meaningful criteria that can be used to calculate dynamic priorities. Though many criteria have been suggested (Reference 1), all tend to be arbitrary and highly dependent

upon the user's objectives and opinions. Therefore, the current version of the ATS uses static priorities to rank the tasks for candidate selection.*

2.2 Identification of Scheduling Opportunities For a Candidate Task

As noted above, a single performance of a task may be described, in part, by a set of resource requirements (allocation requests) which must be met in order to schedule the task. Figure 2.2 illustrates a general set of resource requirements for a task. In the figure, Requirement A might specify the services of a crewman, B a second crewman, C a level of electrical power, etc. As many requirements as needed may be specified.

Each resource requirement exists for a definite interval of time. The length of the interval defines how long the resource is to be allocated to the task while the relative position of the interval defines when, during the task performance, the resource is required. The relative position of the intervals must, therefore, remain fixed. To describe these relations, an arbitrary reference point is selected and designated as the "start-time" of the task. The endpoints of each requirement interval are then assigned values indicating their positions relative to that start-time and hence relative to each of the other intervals. The endpoints may have any desired position relative to the reference point. (Endpoints specified prior to or subsequent to the start-time are equally acceptable.)

Note that the start-time of the task serves as a bridge between the relational time scale shown in the Task Requirement Time Diagram and the actual mission elapsed time scale (MET), for when the start-time of the task is defined in MET, then all of the resource allocations will also be defined in MET.

In addition to these resource requirements, each task has an associated set of performance constraints which must also be met in order to schedule the task. There are two types of constraints: those that define the performance objectives (e.g., minimum number of performances, time

---

*The highest priority task has a priority of 1, the second highest a priority of 2, etc. Tasks with the same priority are selected in the order in which they were input to the program.

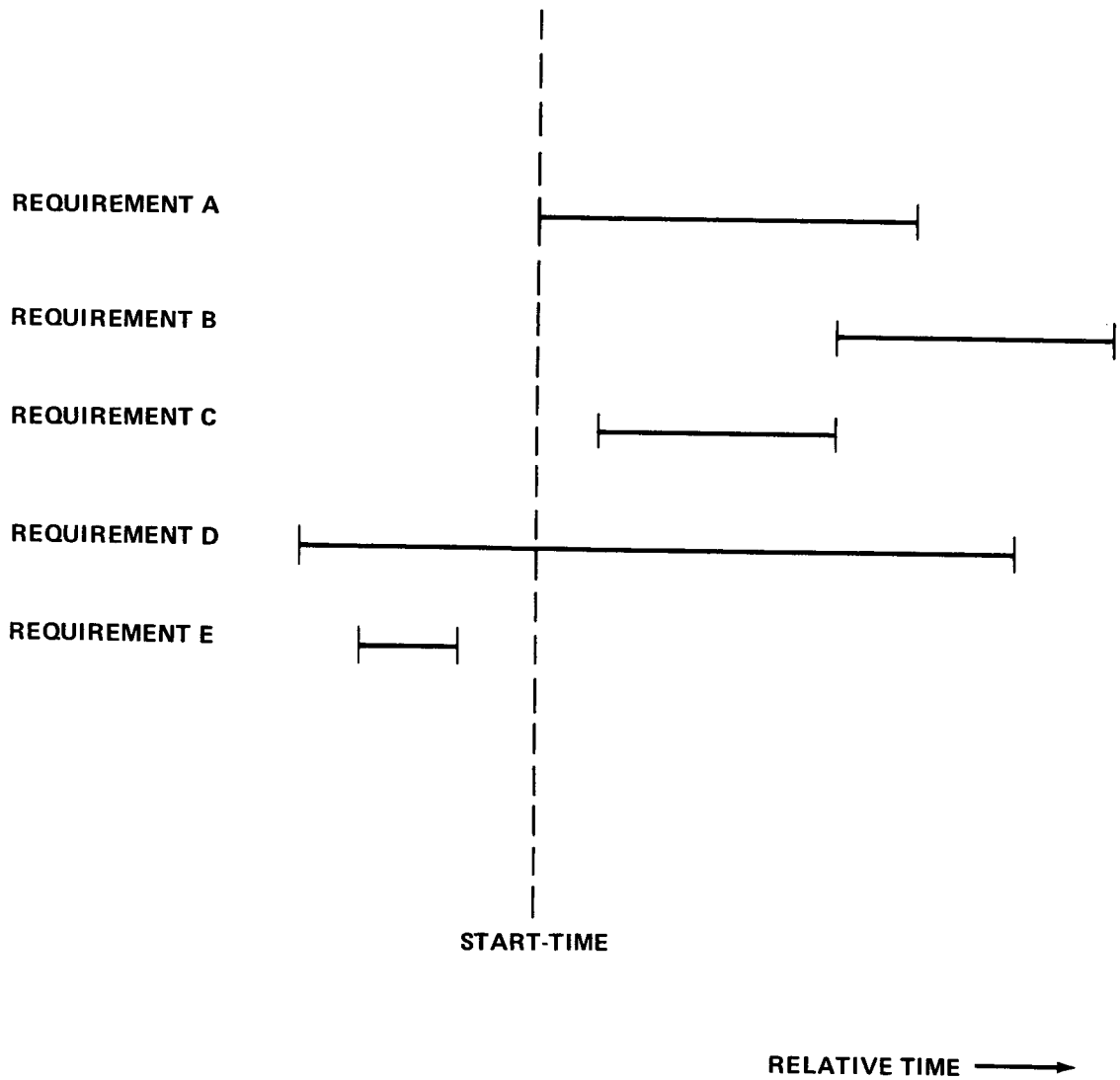


FIGURE 2.2 – TASK REQUIREMENT TIME DIAGRAM.

between performances, etc.) and those which place restrictions on the time of performance (e.g., performance on specific days, or with respect to the performance of another task). Since the ATS uses mission time as the basic independent variable, all resource requirements and performance restrictions are translated into restrictions on the time of performance.

An acceptable interval for the performance of a task can now be defined as an interval of MET during which all resource requirements as well as all performance restrictions can be satisfied. As a corollary, an acceptable start-time for a task is defined as a single value of MET where the task start-time (Figure 2.2) can be placed with the certainty that all of the task's resource requirements and performance restrictions can be satisfied.

The ATS scheduling algorithm finds acceptable start-time windows, i.e., continuous intervals of MET from which a task start-time may be selected for each task. These windows are determined by 'overlaying' the start-time windows determined by considering individual resource requirements and performance restrictions separately. The process is illustrated in Figure 2.3. When only the first requirement, Requirement A, is considered, the task may be initiated at any time within one of the three start-time windows: $A_1 - A_1'$, $A_2 - A_2'$, $A_3 - A_3'$. A range of acceptable start-times for the task based on consideration of Requirement B is then determined for the window $A_1 - A_1'$. In the case shown, the start-time window based only on Requirement B exceeds the limits established by the boundary points $A_1 - A_1'$. Therefore, the boundary points $A_1 - A_1'$ now represent a permissible start-time window based upon consideration of both Requirements A and B. The boundary points of this window are relabeled $B_1 - B_1'$. When Performance Restriction C is considered over the range $B_1 - B_1'$, two separate start-time windows emerge, $C_1 - C_1'$ and $C_2 - C_2'$. The process is repeated again by considering Performance Restriction D over the range of the window $C_1 - C_1'$. When no acceptable start-time windows are found, the window $C_2 - C_2'$ is considered. When the last requirement, Requirement E, is considered over the range $D_1 - D_1'$, the window

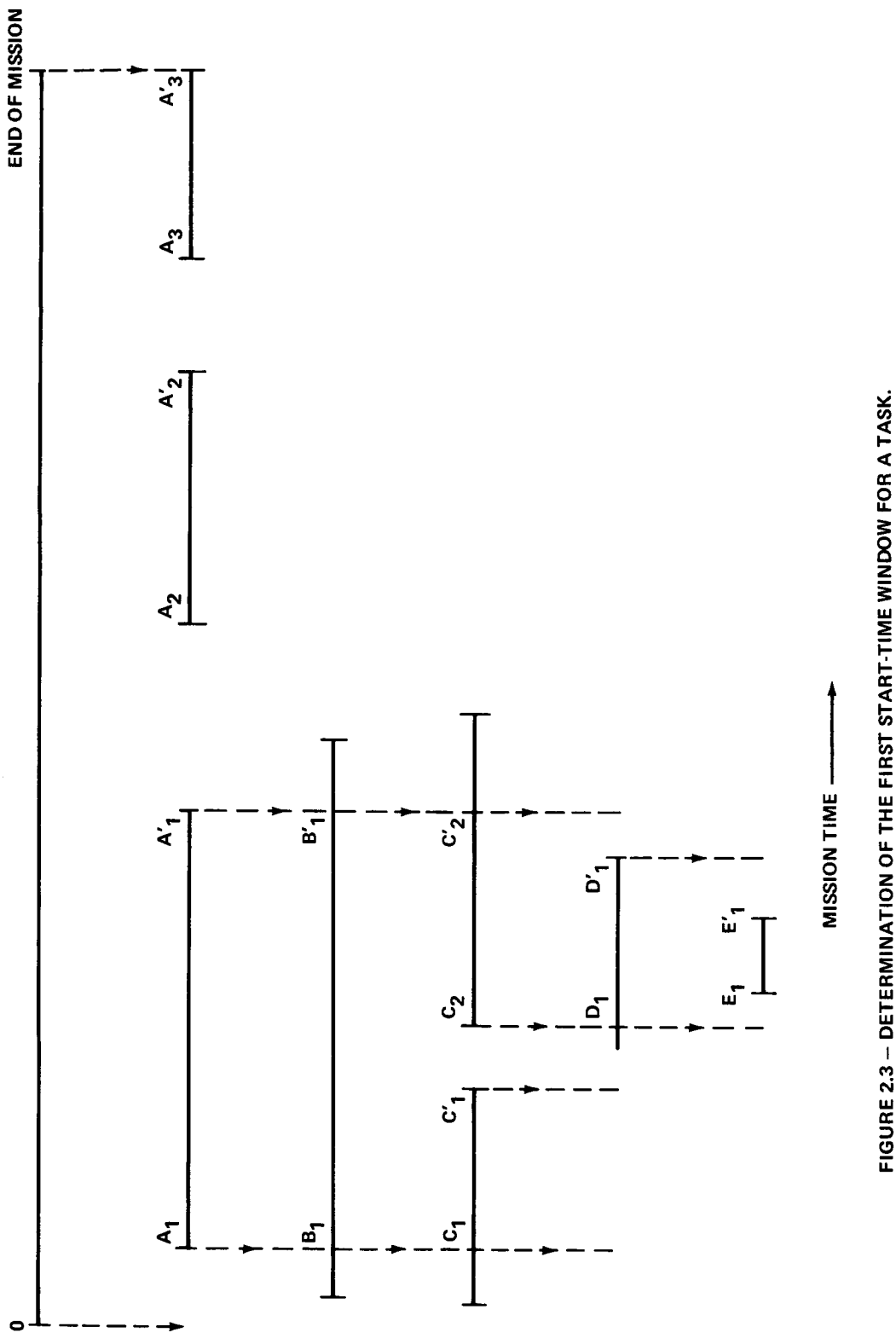


FIGURE 2.3 – DETERMINATION OF THE FIRST START-TIME WINDOW FOR A TASK.

$E_1 - E_1$ is obtained which represents the first acceptable start-time window for the task. If the task is to be performed only once during the mission, the search for start-time windows is terminated when a window is found. If the task is to be performed more than once, the search is continued until all possible windows have been calculated.

2.3 Selection of Task Start-Times

After finding the acceptable task start-time windows, the program selects start-times for as many repetitions of the task as are required, by choosing points within the start-time windows. If start-times can be found for the minimum number of performances required, the task is scheduled, i.e., the timelines of the resources are updated to reflect the performance of the task.

As noted above, the number of scheduling opportunities for any given task is dependent upon commitments already made at the time the task is being considered. Hence, the selection of each task start-time will have some impact on the availability of scheduling opportunities for subsequent candidate tasks. Since no satisfactory method for determining the impact of each selection was apparent, the initial version of the ATS has been programmed to select the earliest possible start-time for the performance of a task. Other alternatives can be evaluated and may be explored in future versions of the system.

3.0 ATS System Description

The ATS System has four primary capabilities. It can

1. Generate a complete schedule (i.e., a set of resource timelines).
2. Complete a partial schedule generated in a previous run.
3. Provide tabular and graphical outputs.
4. Create and maintain a Task Data Bank (i.e., a data library containing descriptions of all the tasks which might be candidates for scheduling on a given mission).

The ATS System flow diagram is shown in Figure 3.1. The System consists of three separate computer programs:

1. A Data Bank Generator that creates and edits a permanent task data bank,
2. A Schedule Generator that generates the resource timelines described in Section 2.3, and
3. A Data Processor that displays the time-line data in graphical form.

The programs are written primarily in FORTRAN V for the UNIVAC 1108 computer. The current version of the ATS is non-conversational (i.e., it must be run in the batch mode) and utilizes externally generated ephemeris data (Section 3.2.1).

3.1 The Data Bank Generator

The Data Bank Generator is a small auxiliary program that is used either to create a new data bank or to edit an existing bank. Task descriptions are input to the program on punched cards in a specified format (Section 4.2). The cards are processed and the final task descriptions are stored on magnetic tape (or FASTRAND file). The output tape, or Data Bank, then contains the latest approved description of the tasks. These include personal tasks (e.g., sleep,

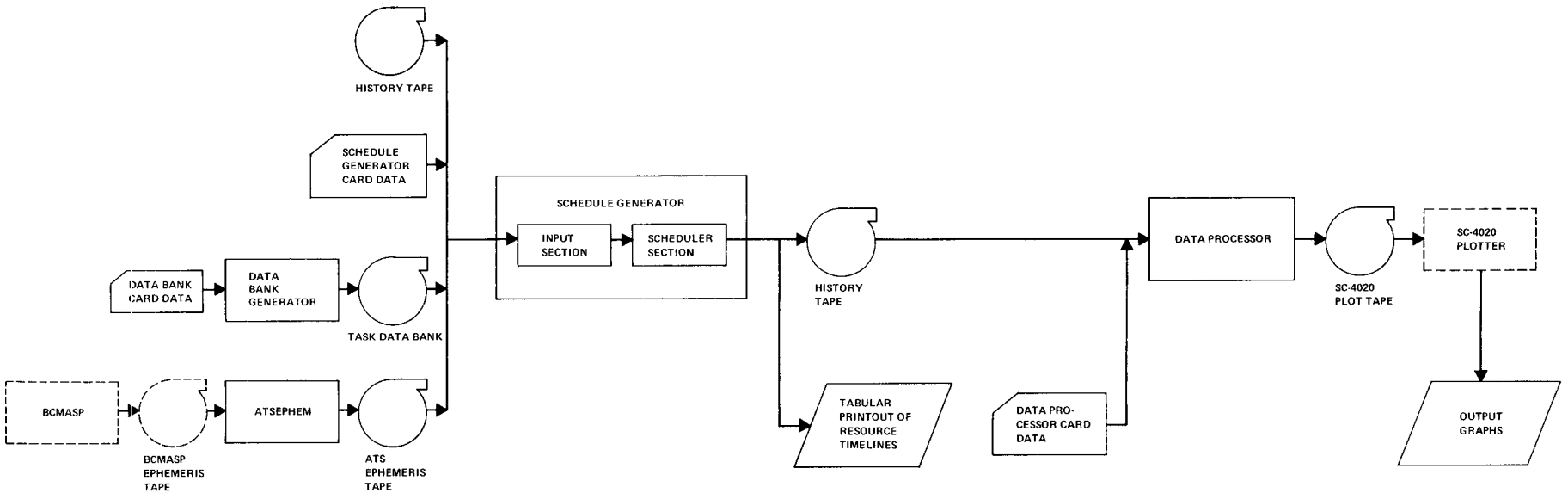


FIGURE 3.1 – ATS SYSTEM FLOW DIAGRAM

breakfast, dinner, etc.), system housekeeping tasks, and scientific experiments. Note that the bank must be maintained and updated as changes to the task descriptions are approved.

3.2 The Schedule Generator

The Schedule Generator is the primary program in the ATS System. It uses the window-filling algorithm described in Section 2.2 to generate a table of commitments (a timeline history) for each of the designated resources. The program has two major subsections: the input section which processes all of the input data and the scheduler section which generates the resource timelines.

3.2.1 Schedule Generator Input Data

As shown in Figure 3.1, there are four sources of input data: the Task Data Bank, the ATS Ephemeris Tape,* card data, and the History Tape.* All sources except card data are optional.

Ephemeris data is generated by a modified version** of the Bellcomm Apollo Simulation Program (BCMASP) and is placed on a magnetic tape in tabular form, each table representing a time history of the availability of the named "resource". Most tables represent line-of-sight contacts between the spacecraft and a celestial or terrestrial "target" such as the sun, an MSFN station, or a photographic objective. Other tables are used to represent the spacecraft's position in its orbit relative to certain cardinal points or areas, e.g. the South Atlantic Anomaly, orbital noon, orbital midnight, etc.

The output tape generated by the BCMASP serves as the input to a small conversion program named ATSEPHEM (Reference 2). The latter selects the tables from the input tape that are to be transmitted to the ATS and writes these tables onto the ATS Ephemeris Tape in a format compatible with the data storage configuration in the ATS (Section 5.1).

---

*These tape files can either be stored on magnetic tape or FASTRAND mass storage.

**Maintenance by the Flight Mechanics Group of Department 1025.

The functional format of an ephemeris table is shown in Table 3.1. The table entries represent intervals of continuous contact so that the spacecraft is shown in contact with the resource during the intervals $t_{11} - t_{12}$, $t_{21} - t_{22}$, etc. As noted above, the ephemeris data need only be input to the Schedule Generator if ephemeris requirements are specified in the descriptions of the tasks to be scheduled. If no ephemeris requirements are specified, the ephemeris tape is not needed to run the Schedule Generator.

Four types of card data are input to the Schedule Generator: mission characteristics, system constraints, program control cards, and task description data. The first three types are input in the first section of the data deck in free format. Task description data cards are input in the second section.

The user can, if he so desires, input all of the task descriptions directly to the Schedule Generator from punched cards. The descriptions are stored on peripheral drum storage for the duration of the run. These temporary files are the sole source of task description data used by the Schedule Generator. If the bulk of this data does not vary from run to run, the user may alternately assign the Data Bank as an input source and designate, via the program control cards, which of the task descriptions on the bank are to be used for that particular run. The specified tasks are then copied from the Data Bank onto the temporary files for the duration of the run.

The same task description cards are used to edit any of the task descriptions stored on the drum files as well as to add new tasks for the duration of the run. To provide the capability of manipulating the task descriptions, a Task Description Language (TDL), used for specifying task requirements and performance constraints, was developed for the ATS. The language consists of 12 standard card formats. When used appropriately, it enables the user to perform all of the functions described above (i.e., to add, delete, and edit task descriptions on the temporary files). The same language and input structure are used for the Schedule Generator and the Data Bank Generator. Details of the TDL and the associated editing procedures are presented in Section 4.0.

3.2.2 Schedule Generator Output

The primary output of the Schedule Generator is a set of resource commitment tables and a list of start-times

Table 3.1

Ephemeris Resource Table*

t_{11}	t_{12}
t_{21}	t_{22}
⋮	
t_{n1}	t_{n2}

*$t_{11} < t_{12} < t_{21} < t_{22} < \ldots < t_{n1} < t_{n2}$

for each task. At the beginning of the scheduling process, all of these tables are empty since no commitments have been made. Subsequently, whenever a task is scheduled, an entry is made in the appropriate tables to reflect the commitment of each resource to the task for the amount of time specified in the Task Requirement Time Diagram (Figure 2.2). Therefore, at any point in the scheduling process, the commitment tables contain an up-to-date history of the resource allocations.

All resources can be classified as either binary or analog. A binary resource has only two possible states: committed or uncommitted. Therefore, when a binary resource is allocated to an activity or task for a specified time interval, it is considered unavailable for any other assignment over that interval.* Examples of binary resources include crewmen, pieces of equipment, or (the occupancy of) a scientific airlock. The configuration of a binary resource commitment table is shown in Table 3.2a. In the table, the resource is shown committed to Task #1 over the interval $t_{11} - t_{12}$, to Task #2 over the interval $t_{21} - t_{22}$, etc.

In contrast to a binary resource, requirements on an analog resource (e.g., power, oxygen, water, etc.) are specified quantitatively. An analog resource may therefore be simultaneously allocated to any number of tasks as long as the sum of all the allocations does not exceed the specified maximum. For example, assume that a particular power source can deliver a maximum of 1000 watts. Then, the only limitation on the usage of that power source is the 1000 watt maximum. Any number of tasks may use the power source simultaneously so long as the total power consumption does not exceed 1000 watts. The configuration of an analog resource commitment table is shown in Table 3.2b. The table records the total allocation of the resource over a specified interval; hence, R_{12} represents the total commitment over the interval $t_{11} - t_{21}$, R_{22} the total over the interval $t_{21} - t_{31}$, etc.

---

*An ephemeris resource is a type of binary resource since it has only two states: available and unavailable. However, ephemeris resources are excepted from the single allocation rule. Thus, an ephemeris resource can be allocated to any number of tasks over the same period of availability.

Table 3.2 Commitment Table Configuration*

	Column #1	Column #2	Column #3
a. Binary Resource Table**			
	t_{11}	t_{12}	Task #1
	t_{21}	t_{22}	Task #2
	⋮		
	t_{n1}	t_{n2}	Task #n
b. Analog Resource Table†			
	t_{11}	R_{12}	
	t_{21}	R_{22}	
	⋮		
	$t_{(n-1)1}$	$R_{(n-1)2}$	
	t_{n1}	0	

*t_{ij} are values of mission elapsed time.

**$t_{11} < t_{12} \leq t_{22} \leq \cdots \leq t_{n1} < t_{n2}$.

†$t_{11} < t_{21} < \ldots < t_{(n-1)1} < t_{n1}$.

All of the resource commitment tables are printed out at the completion of the schedule, and on option, at regular intervals during the scheduling process. Also on option, the complete set of resource tables will be written onto the History Tape at each priority level. Each of the n sets of records on the tape will therefore contain all of the information necessary to define the status of the resource timelines at a particular point in the scheduling process.

The History Tape has two uses: to serve as an input to the Data Processor and to serve as an input to the Schedule Generator itself on a subsequent run. The second use enables the user to initialize all of the resource tables to their original status at some intermediate priority (i) by reading in the appropriate set of records from the History Tape. The scheduling process would then begin with the tasks having a priority of i+1. This option enables the user to modify an existing schedule as well as to generate a completely new one. The option is particularly important because it can be used to conduct economical (in computer time and charges) investigations of variations in a basically desirable schedule.

3.3 The Data Processor

The ATS Data Processor is used to graphically display the timeline data produced by the Schedule Generator. The program has two sources of input data: punched cards and the History Tape. The card data contains program control instructions, the names of the variables to be plotted, and the manner and the scale to which they are to be plotted. Timeline data is obtained directly from the History Tape.

The output of the Data Processor is a magnetic tape which contains specific instructions for the Stromberg-Carlson SC-4020 plotter. The latter generates graphs by photographing the sequential displays of a cathode-ray tube. Hence, the graphical information must be generated frame-by-frame.

The current version of the Data Processor can generate two types of plots: "coaxial" and "periodic". The coaxial plots can display up to five variables on a single set of axes. The variables may be analog resources, binary resources, ephemeris resources, task performances, or any combination of these. Two examples of coaxial plots are shown in Figure 3.2. In both plots, the abscissa represents MET measured from the SL-2 insertion on a Skylab mission. Figure 3.2a shows a portion of the timelines for three crewmen. The data, taken directly from the binary

COAXIAL PLOTS OF CREW TIMELINES

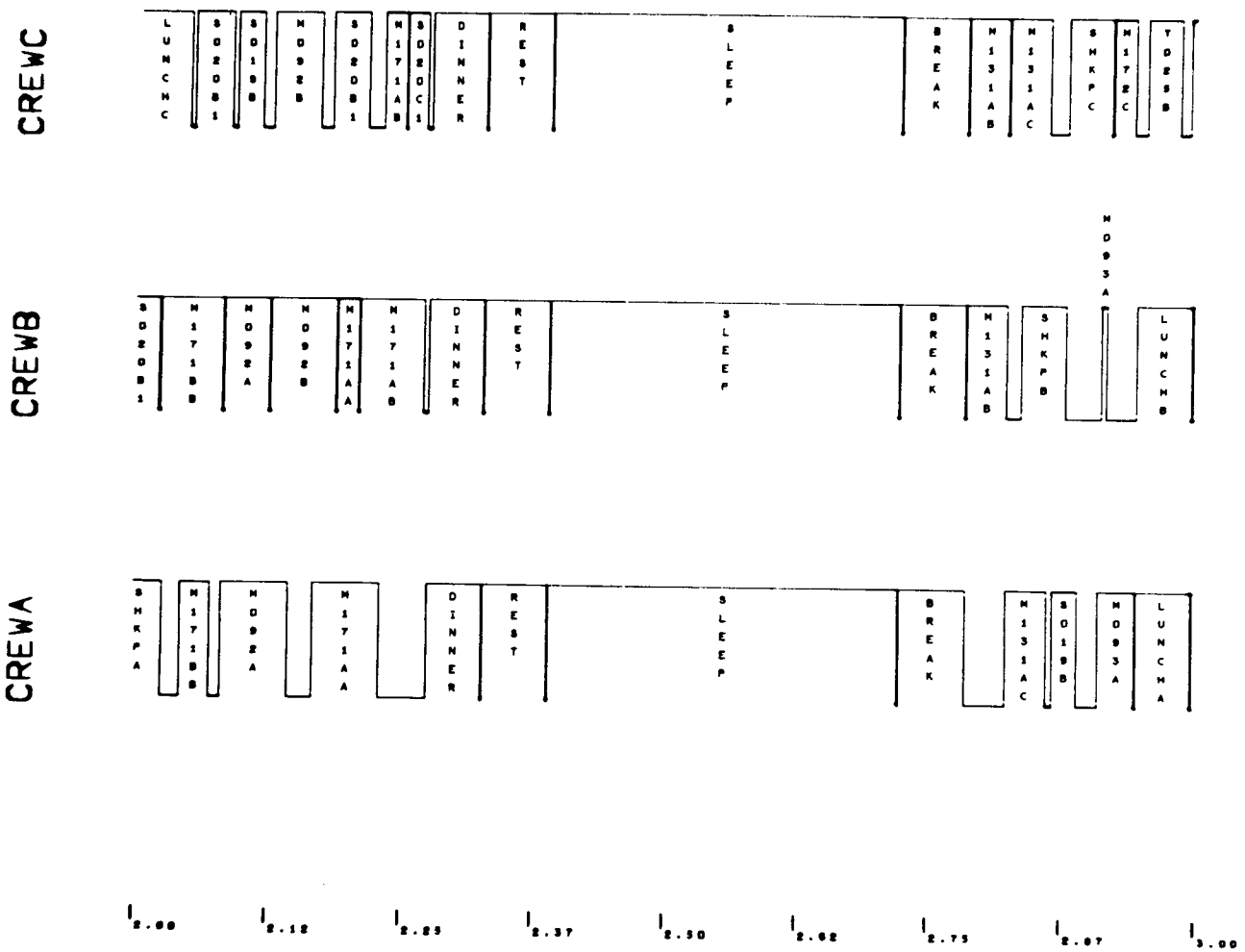


FIGURE 3.2a

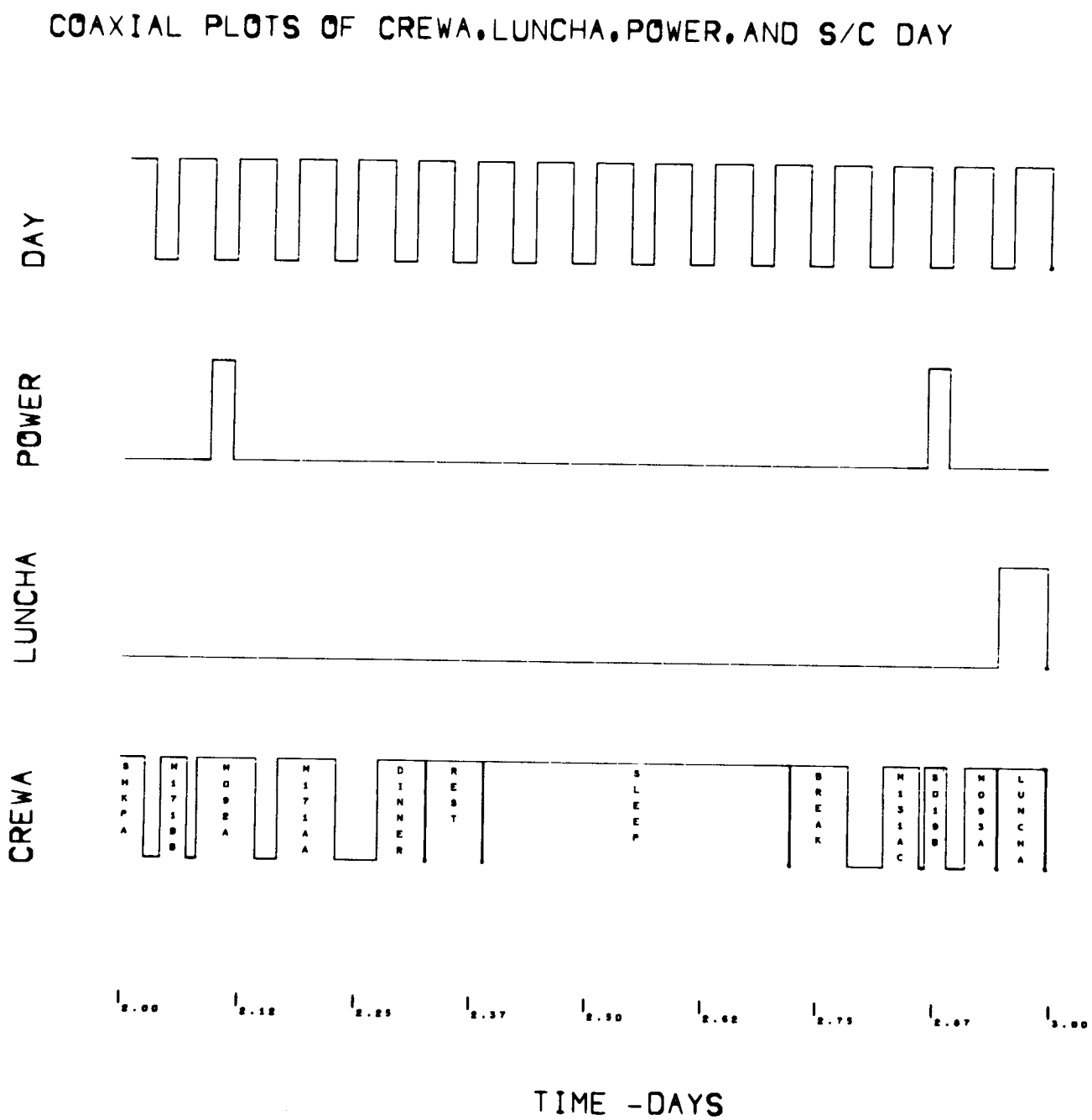

COAXIAL PLOTS OF CREWA, LUNCHA, POWER, AND S/C DAY
DAY
POWER
LUNCHA
CREWA
SKPA
M171B
M09A
M171A
DINNER
REST
SLEEP
BREAK
M151AC
BOB
M09A
LUNCHA
I2.00
I2.12
I2.25
I2.37
I2.50
I2.62
I2.75
I2.87
I3.00
TIME -DAYS

FIGURE 3.2b

commitment tables for each crewman, shows the specific task associated with each interval of committed time. Figure 3.2b illustrates the different types of data that can be plotted on one coaxial plot: CREWA is again the graphical representation of the crew timeline for Crewman CREWA, LUNCHA shows the occurrences of the Task LUNCHA (the lunch period for Crewman CREWA), POWER is an analog resource representing the total electrical power required by the scheduled activities, and DAY, an ephemeris resource, shows the times that the spacecraft is within line-of-sight of the sun.

The periodic plots provide the capability to overlay data from resource commitment tables, ephemeris commitment tables, and task performance times in order to observe recurring patterns and the interrelationships between the different variables. On the periodic plots, the occurrences can be plotted as shaded boxes or as points (the point representing the midpoint of the occurrence). A portion of a periodic plot is shown in Figure 3.3. Figure 3.3a shows the plot legend, printed at the beginning of every plot. Figure 3.3b shows the plot itself. The symbol for DAY represents the times the spacecraft is in sunlight while the symbols for REST and LUNCHA represent performances of tasks by the same name.

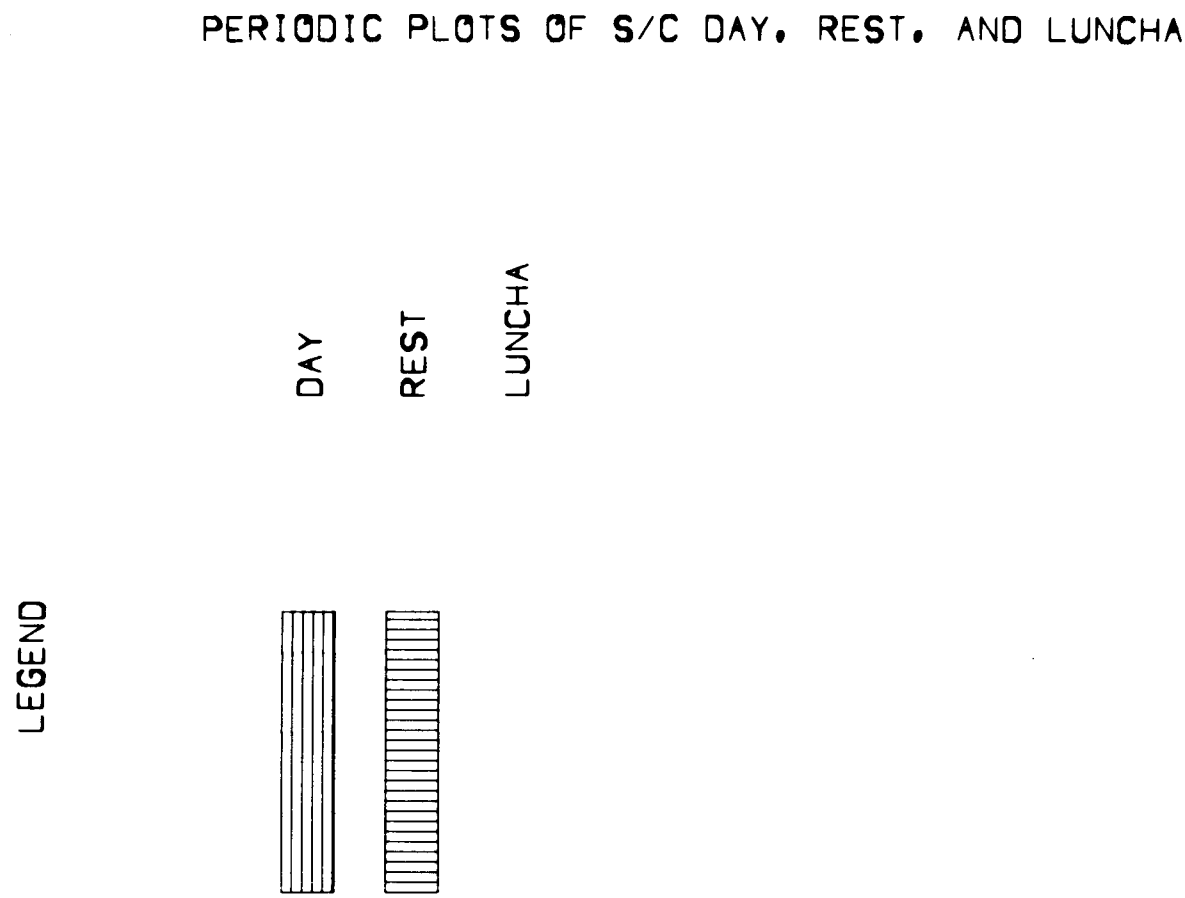


FIGURE 3.3a – LEGEND FOR A PERIODIC PLOT.

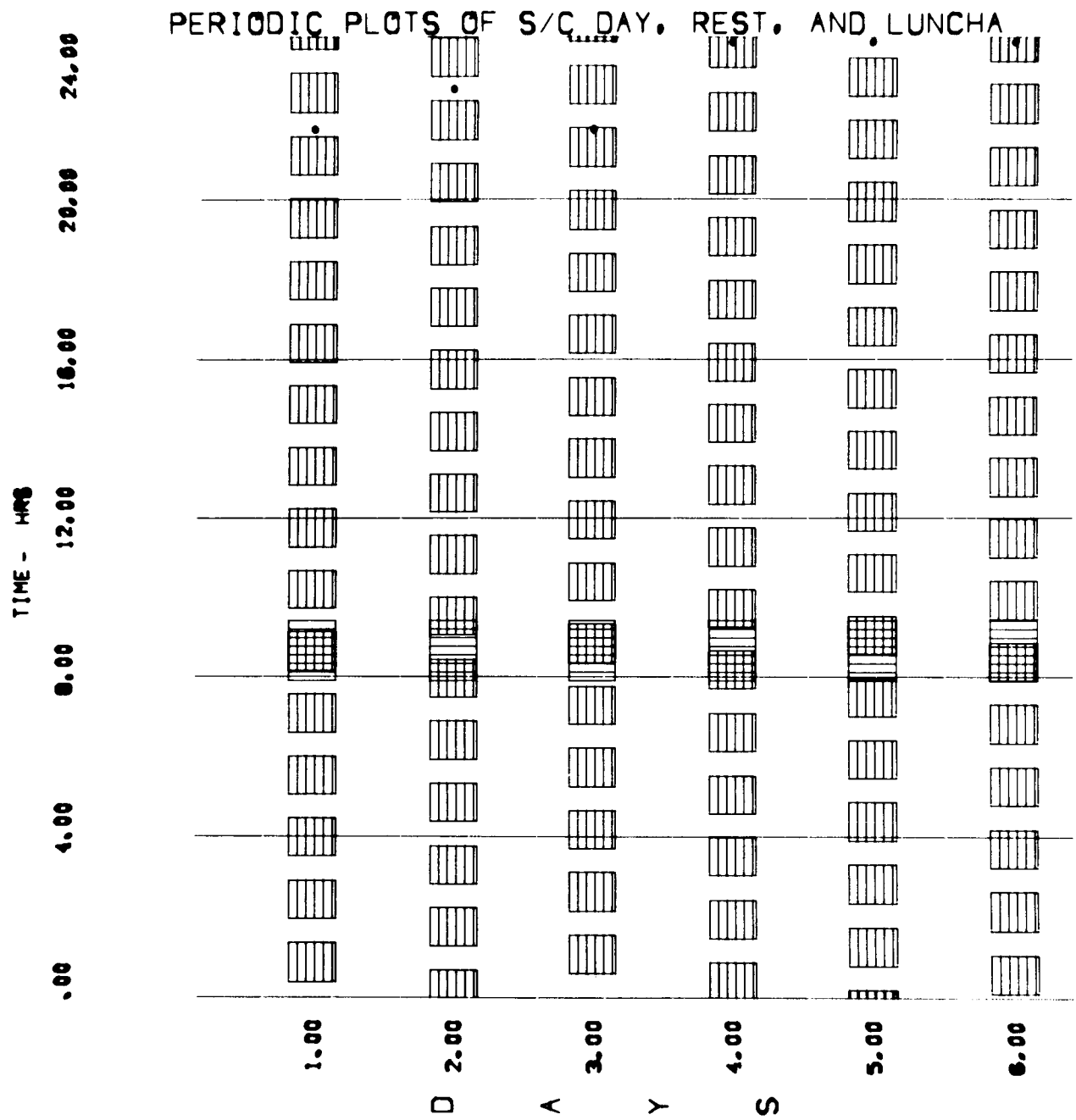


FIGURE 3.3b – PORTION OF A PERIODIC PLOT.

4.0 The ATS Task Description Language

4.1 Language Structure

An input language was developed for the ATS to translate task requirements and performance constraints into statements that could be used by the ATS programs.* The primary requirement of the input language is that it have sufficient capability to interpret all task specifications, despite the fact that the nature of these specifications may vary markedly from task to task. To provide this flexibility, the language structure contains a common base within which all task specifications must be described. The common base is composed of the seven descriptors defined in Table 4.1. Resource requirements and performance constraints for all tasks must be translated by the user into three or more statements using these descriptors. The descriptors will be discussed more fully in Section 4.2.

4.2 Card Formats

The current version of the Task Description Language (TDL) consists of the 12 card formats shown in Table 4.2. The formats are divided into three groups:

1. Descriptor Cards which are used to specify a task's performance objectives, resource requirements, and performance restrictions.

2. Annotation Cards which are used to annotate the Descriptor Cards with any alphanumeric information desired by the user. (The Annotation Cards are available as a convenience and do not affect the scheduling of the task in any way.)

---

*The Task Description Language is used by both the Data Bank Generator and the Schedule Generator. Usage rules apply to both programs.

Table 4.1

TASK DESCRIPTORS

Descriptor	Purpose
PRIORITY	Specifies when, relative to all other tasks, this task will be considered for scheduling.
OBJECTIVE	Specifies the number of task performances and the spacing between performances.
TIME	Specifies the permissible intervals of MET during which the task may be initiated.
ENABLE	Specifies the permissible interval, relative to the start-time of another task, during which the task may be initiated.
INHIBIT	Specifies the interval, relative to the start-time of another task, during which the task may not be initiated.
AMOUNT	Specifies the total amount of a consumable required for one performance of the task.
RESOURCE	Specifies the time interval, relative to the start-time of the task during which the resource is required.

TABLE 4.2 ATS TASK DESCRIPTION CARDS

	CARD NAME	IDENTIFIER FIELDS	DATA FIELDS			
		1 2	3	4	5	6
DESCRIPTOR CARDS						
	PRIORITY CARD	TASK / PRI,	K			
	OBJECTIVE CARD*	TASK / OBJEC,	NPERFQ,	DPERF,	$t_{bet'}$	t_{tol}
	TIME CARD*	TASK / TIME,	t_1,	t_2,	t.....	
	AMOUNT CARD	TASK / AMOUNT,	RNAME,	A		
	ENABLE CARD*	TASK / ENABLE,	TNAME,	t_1,	t_2,	t....
	INHIBIT CARD*	TASK / INHIB,	TNAME,	t_1,	t_2,	
	RESOURCE CARD*	TASK / RES,	RNAME,	t_1,	t_2,	R
ANNOTATION CARDS						
	TITLE CARD	TASK / TITLE,	K			
	COMMENT CARD	TASK / COMMNT,	K			
INSTRUCTION CARDS						
	EQUIVALENCE CARD	TASK / EQUIV,	TNAME			
	DELETE CARD	TASK / DELETE				
	LAST CARD	LAST				

*ALL VALUES OF TIME, DENOTED BY t, ARE EXPRESSED IN INTEGER VALUES OF DAYS, HOURS, AND MINUTES IN THE GENERAL FORMAT DD:HH:MM.

3. Instruction Cards which are used to provide specific instructions to the program on the processing of the task descriptions.

Each card (except the Last Card) has two sets of fields: identifier fields and data fields. The first identifier field on each card identifies the name of the task. The latter may be any combination of letters and numbers up to a maximum of six characters. The first character, however, must be a letter. The second field on each card identifies the card type which in turn determines how the information on the card is to be processed. The 12 card types shown in Table 4.2 are the only types permitted in the present version of the TDL and the type designation must appear on the punched cards exactly as shown. The remaining identifier and data fields are unique to the particular card type and will be discussed below.

4.2.1 Descriptor Cards

Each set of task Descriptor Cards must contain one Priority Card and one Objective Card. The remainder of the set can be made up of any desired combination (quantity and type) of the remaining five card types.

The card formats are structurally similar. Each card may be submitted in free format (i.e., independent of position). Blank columns will be ignored. However, the fields must appear in the order specified and a delimiter must be used between successive fields. The delimiter may be either a slash (/) or a comma. If all of the information is included on one card, no delimiter should be placed after the last field on the card. However, if more than one card is needed, then a comma should be placed after the last field on the card and the information continued on another card. Note that these supplementary cards must then appear in the data deck together and in the proper sequence.

4.2.1.1 Priority Card

The third field of the Priority Card contains an integer k, the numerical priority assigned the task ($k \geq 1$). During the generation of the schedule, the tasks are considered for scheduling in order of their numerical priority (e.g., the first task considered has a priority of one). If two or more tasks have the same numerical priority, they will be considered for scheduling in the order in which they were input to the program.

4.2.1.2 Objective Card

The task's repetition requirements are specified on fields 3 through 6 of the Objective Card. In Field 3, variable NPERFQ is an integer defining the minimum number of required task performances (NPERFQ≥1). If this minimum cannot be scheduled, then no performance of the task is scheduled. In Field 4, variable DPERF is an integer defining the maximum desired number of task performances (DPERF≥NPERFQ). The Schedule Generator will schedule as many performances as possible between the specified minimum and maximum. If only one performance of a task is desired, then both NPERFQ and DPERF should be set equal to one. Note that the maximum value of DPERF is limited by the first dimension of an internal working array (Array LVWIN) in the Schedule Generator. In the current version of the ATS, the array is dimensioned (60,3). Thus, DPERF cannot be greater than 60.

Fields 5 and 6 need only be specified for multi-performance tasks (DPERF≥1). The variable t_{bet} in Field 5 specifies the nominal time interval between successive performances while t_{tol} in Field 6 specifies the tolerance on that nominal value. As an alternative, the user may designate t_{bet} as the minimum time between successive performances by specifying the letters MIN in Field 6. When Fields 5 and 6 are not supplied, the performances will be scheduled without regard to the spacing between repetitions.

4.2.1.3 Time Card

Unless otherwise specified, the Schedule Generator assumes that the task may be initiated at any time over the mission duration where the resource requirements and performance constraints are satisfied. The Time Card permits the user a degree of control over the scheduling of a task by defining specific intervals of mission elapsed time as acceptable start-times. These continuous intervals are considered as additional restrictions on the performance of a task when deriving acceptable task start-time windows and so, by the process described in Section 2.3, the task start-time will always be defined within the specified interval. Note that the Time Card can be used to insert a task performance at any particular value of MET by setting both endpoints of the interval to that value. The performance will be scheduled at that point provided all requirements and performance constraints can be satisfied.

For the Time Card shown in Table 4.2, the values of t_1 and t_2 in Fields 3 and 4 represent the lower and upper endpoints respectively of the acceptable interval ($t_2 \geq t_1$). The endpoints are values of mission elapsed time and must always be specified in the order shown. As indicated by the dots, any number of intervals may be specified on one card (or its continuation). For example, the values of t_3 and t_4, representing endpoints of a second acceptable interval ($t_4 \geq t_3$), could be added to the card in Fields 5 and 6, etc. The endpoints must be specified in pairs with one exception: the second endpoint of the last interval on the card may be omitted leaving the interval open-ended. In this case, the Schedule Generator assumes the interval ends at the end of the mission.

4.2.1.4 Amount Card

The Amount Card is used to specify that a quantity A of consumable RNAME is required for each performance of the task. A given quantity of each consumable is allocated for the mission. When each task is scheduled, this total is diminished by an amount equal to the product of quantity A and the number of performances scheduled. Therefore, at any point in the scheduling process the amount of the consumable still uncommitted is known. When a task is considered for scheduling, the maximum number of performances permitted will be limited to the number that would require no more of the consumable than is currently available. If that number is less than the required minimum (variable NPERFQ on the Objective Card) the task is not scheduled.

4.2.1.5 Enable Card

The Enable Card specifies intervals of acceptable task start-times relative to the start-time of another task. The latter, considered the independent task, is identified in Field 3 while the task to which the constraint applies (i.e., the task in Field 1) is designated the dependent task. Implicit in the specification of an ENABLE constraint is that performances of the independent task have already been scheduled. If no performances have been scheduled, the ENABLE constraint cannot be satisfied and the dependent task is not scheduled. The values t_1 and t_2 in Fields 4 and 5 are the endpoints of an acceptable start-time interval relative to the start-time of each performance of the independent task. The values of t_1 and t_2 may be positive or negative as long as $t_2 \geq t_1$.

As indicated by the dots, the Enable Card also permits any number of intervals to be specified on one card (or its continuation). As with the Time Card, the endpoints must be specified in pairs with the exception of the last interval. The second endpoint of the last interval may be omitted, leaving the interval open-ended, in which case the Schedule Generator assumes the interval ends at the end of the mission.

As mentioned above, all intervals defined on the Enable Card apply to every performance of the independent task; i.e., the acceptable start-time windows are defined relative to every performance of the independent task. However, the user may designate that the intervals on the card are to apply only to the latest scheduled performance of the independent task by inserting the symbols (LAST) before the name of the independent task in Field 3. In this case the card becomes

TASK/ENABLE, (LAST) TNAME, t_1, t_2

Note that all independent tasks must be considered for scheduling before the dependent task is considered. If the dependent task is considered before any of the independent tasks, the run will terminate with an error message (Appendix A - Section A.2.3.3).

4.2.1.6 The Inhibit Card

The function of the Inhibit Card is the inverse of the function of the Enable Card; the Inhibit Card specifies an unacceptable start-time interval relative to the start-time of another task. As in Section 4.2.1.5, the name of the independent task appears in Field 3 and the values t_1 and t_2 in Fields 4 and 5 represent values of time relative to the start-time of each performance of the independent task. However, these values now represent the endpoints of an interval in which no performance of the dependent task can be initiated. As above, the values of t_1 and t_2 may be positive or negative so long as $t_2 \geq t_1$.

The interval defined on an Inhibit Card applies to every performance of the independent task without exception. In contrast to the Enable Card, only one interval can

be specified on an Inhibit Card and both endpoints of this interval must be defined. In addition, all independent tasks must be considered for scheduling before the dependent task is considered. If the tasks were considered in reverse order (dependent task before the independent task), it would be possible to schedule both tasks in violation of the inhibit requirement. Therefore, if the dependent task should be considered before any of the designated independent tasks, the run will terminate with an error message (Appendix A - Section A.2.3.3).

4.2.1.7 The Resource Card

The Resource Card is used to state a requirement that a given resource be available for a specified interval relative to the start-time of the task. A separate card must be used for each requirement and only one pair of endpoints (i.e., one continuous interval) may be specified on a card. As shown in Table 4.2, the name of the resource is specified in Field 3 and the endpoints of the required interval in Fields 4 and 5. The values of t_1 and t_2 represent the earliest and latest requirement times and, as shown in Figure 2.2, may be either positive or negative depending upon their position relative to the start-time of the task.

The Resource Card may be used to specify requirements on any type of resource (e.g., binary, analog, or ephemeris). For the binary and ephemeris resources, specification of the endpoints t_1 and t_2 is sufficient and Field 6 is left blank. For an analog resource however, Field 6 must contain a number indicating the quantity of the resource required over the interval. The units of this number should be the same as the units in which the maximum permissible value was specified.

The name of the resource specified in Field 3 must be identical to the name of the corresponding resource table stored in the program. These names may be any combination of letters and numbers up to a maximum of six, however, the first and last characters of the name must be a letter. There are three exceptions to the rule that the name in Field 3 be identical to the resource table name. They are described below.

4.2.1.7.1 Designation of Multiple Requirements on the Same Resource

The first exception is designed to provide for cases in which there is more than one requirement on the same resource in the same task description. If, for example, Requirements B and C in Figure 2.2 represent two different

power levels on the same resource, POWER, then a number would be affixed to each resource name in Field 3 to distinguish between the two requirements.* The Resource Cards for these two requirements would then become

Requirement B TASK/RES, POWER1, t_1, t_2, R_B

Requirement C TASK/RES, POWER2, t_1, t_2, R_C

During the execution of the program, the final numeral is ignored and the resource is identified by its table name, POWER. Note therefore that resources for which multiple requirements are defined must be assigned a name containing no more than five characters so that the identifying numeral may be affixed where necessary.

4.2.1.7.2 Crewman Designation

The second exception to the name rule is designed to enable the user to specify participation of a crewman other than by name. On option, a specific skill may be assigned to any or all of the crewmen.** The third identifier field may then contain the name of a skill (any combination of letters or numbers to a maximum of six characters so long as the first character is a letter) rather than the name of a specific crewman. The program will select the crewman by the designated skill rather than by name. If neither identity nor skill is important, the special designation ANY should be placed in Field 3. This designation permits the Schedule Generator to select any crewman whose availability is consistent with the requirements of the task.

When the ANY designation appears in Field 3, the crewmen are considered for selection in inverse order of total committed time. The selection of a crewman for each window at that requirement level is independent of selections made

---

*The necessity to distinguish between these two requirements will be explained in Section 4.4.

**The current version of the ATS permits only one skill to be assigned to each crewman. No two crewmen may be assigned the same skill.

for other windows at the same level. If, for example, Requirement A in Figure 2.3 represents the requirement for an undesignated crewman, the selection of a crewman for each of windows $A_1 - A_1'$, $A_2 - A_2'$, $A_3 - A_3'$ would be independent of the selection made for the other two. The option does not therefore permit the user any control over the selection of the crewman.

4.2.1.7.3 Inverse Designation for Ephemeris Resources

The third exception to the name rule permits the user to specify that the spacecraft be out of contact with any desired ephemeris resource. For example, the ephemeris resource table for the South Atlantic Anomaly (named SAA) would contain the intervals during which the spacecraft is in contact with the SAA. If the task must be performed outside the SAA, the third field of the Resource Card should contain the designation (NOT)SAA. The Resource Card for that option would become

TASK/RES, (NOT)SAA, t_1, t_2

4.2.2 Annotation Cards

The Annotation Cards are designed to permit the user to annotate the task description cards with alphanumeric information. There are two types of cards, Title Cards and Comment Cards; both are used in an identical manner. The number of each type used is entirely optional with one exception; each task must be introduced by a Title Card. The cards of each type must be numbered consecutively. This number appears in the third field of the card (shown as k in Table 4.2) and is always followed by a blank space. All information appearing on the card after the blank space is interpreted as alphanumeric information. As such, it is not processed by the Schedule Generator Input Section but is stored exactly as it appears on the card.

4.2.3 Instruction Cards

The three instruction cards shown in Table 4.2 are used to issue specific instructions to the programs. They are never used as part of a set of task specifications.

4.2.3.1 The Equivalence Card

The Equivalence Card permits the user to duplicate a task description already in storage. It provides a convenient alternative to introducing a second set of task

description cards that are virtually identical to a set already in storage. When encountering an Equivalence Card, the program duplicates all of the Descriptor and Annotation Cards for the task named in Field 3 and associates the duplicate set with the new task named in Field 1. After this card is processed, there will be two separate, distinct, and identical sets of cards in storage. The duplicate set should then be modified (Section 4.5) to obtain the exact description required for the new task. Note that further modifications to either set of cards will not affect the other set.

4.2.3.2 The Delete Card

The Delete Card is used to delete the task named in Field 1 from the permanent data bank. All Descriptor and Annotation Cards pertaining to the task are deleted. This card is only used as an input to the Data Bank Generator.

4.2.3.3 The Last Card

The Last Card has the word LAST in the first identifier field (Field 1). No other fields appear on the card. The card is placed after the last Descriptor or Annotation Card in the data deck to indicate to the program that all of the task definitions and modifications have been specified.

4.3 Generating a New Task Description

There are two methods of entering task descriptions into an ATS program: inputting the entire set of Annotation and Descriptor Cards that define the task or using an Equivalence Card to duplicate the description of a task already stored.

4.3.1 Translation of Task Specifications Into a Set of Task Description Cards

When a new set of cards is used to define a task, the following rules governing the input sequence must be observed:

1. A Title Card must be used to introduce the name of the new task. Therefore, the first Title Card must be input before any other card in the set.

2. The Priority and Objective Cards must be input (in that order) before any other Descriptor Cards.

3. The remaining Descriptor Cards may be placed in any desired order. They will be stored in the same order they are input.

These rules are demonstrated in the Task Description Cards for two illustrative tasks named Sleep and Breakfast shown in Figures 4.1a and 4.1b, respectively. The cards in each set are shown in the order in which they would be input.

A more sophisticated illustration of the translation of task requirements and performance constraints into the required card formats is obtained from examining Task M093, Vectorcardiogram, one of the in-flight experiments to be performed in the Skylab Program. The task is designed

> "to measure electrocardiographic potentials of each astronaut during the weightless period and the immediate post-flight period by methods that will allow precise quantitative measurement of the changes that occur. The experiment is conducted on each crewman every third day during the mission by obtaining vectorcardiogram measurements at rest and while exercising on an ergometer."*

The measurements are made by attaching electrodes to different parts of the subject's body. The output signals from these electrodes are processed by an auxiliary piece of electronic equipment (the Experiment Support System or ESS) and recorded for future transmission to earth.

The characteristics of Task M093 that are significant to scheduling are shown in Figure 4.2. The Task Timeline Diagram for one performance of the task is derived from this data and is shown in Figure 4.3. Finally, Figure 4.4 shows the set of Task Description Cards for one performance of M093A (M093 for one subject designated Crewman A) derived from the Task Timeline Diagram and the operational constraints given in Figure 4.2.

---

*Reference 3.

Figure 4.1a

Task Description Cards for the Sleep Task

```
SLEEP/TITLE,      1                          TASK SLEEP
SLEEP/COMMNT,     1          ALL CREWMEN SLEEP SIMULTANEOUSLY FOR ONE
SLEEP/COMMNT,     2          CONTINUOUS 8-HOUR PERIOD EVERY 24 HOURS.
SLEEP/COMMNT,     3          THE FIRST PERIOD SHOULD NOT BEGIN PRIOR
SLEEP/COMMNT,     4          TO 00:09:25 MISSION ELAPSED TIME.
SLEEP/PRI,        1
SLEEP/OBJEC,      27, 27,    01:00:00,     00:00:00
SLEEP/TIME,       00:09:25
SLEEP/RES,        CREWA,     00:00:00,     00:08:00
SLEEP/RES,        CREWB,     00:00:00,     00:08:00
SLEEP/RES,        CREWC,     00:00:00,     00:08:00
```

Figure 4.1b

Task Description Cards for the Breakfast Task

```
BREAK/TITLE,      1                          TASK BREAK
BREAK/COMMNT,     1      CREWMEN EAT BREAKFAST TOGETHER IMMEDIATELY AFTER
BREAK/COMMNT,     2      AWAKENING FROM SLEEP.  ONE HOUR AND 30 MINUTES
BREAK/COMMNT,     3      ARE ALLOTTED FOR BREAKFAST.
BREAK/PRI,        2
BREAK/OBJEC,      27, 27,    01:00:00,     00:00:00
BREAK/ENABLE,     SLEEP,     00:08:00,     00:08:00
BREAK/RES,        CREWA,     00:00:00,     00:01:30
BREAK/RES,        CREWB,     00:00:00,     00:01:30
BREAK/RES,        CREWC,     00:00:00,     00:01:30
```

Figure 4.2

SCHEDULING CHARACTERISTICS OF TASK M093 - VECTORCARDIOGRAM

TASK DESCRIPTION

OBJECTIVE - ONE PERFORMANCE ON EACH CREWMAN ONCE EVERY THIRD DAY IN FLIGHT.

CREW REQUIREMENTS

SUBJECT	0 - 39 MINUTES
OBSERVER	0 - 39 MINUTES

EQUIPMENT REQUIREMENTS

BICYCLE ERGOMETER (ALSO USED BY TASK M171 - METABOLIC ACTIVITY)

VECTORCARDIOGRAM VEST (ALSO USED BY TASK M092 - IN-FLIGHT LOWER BODY NEGATIVE PRESSURE)

EXPERIMENT SUPPORT SYSTEM (ALSO USED BY M092, M171, M131)

POWER REQUIREMENTS

10 WATTS FOR THE FIRST 32 MINUTES

ADDITIONAL 15 WATTS FOR THE TWO MINUTE EXERCISE ON THE ERGOMETER. THE EXERCISE IS PERFORMED DURING THE 20 AND 21ST MINUTE OF THE CREWMEN'S PARTICIPATION.

OPERATIONAL CONSTRAINTS

NO PERFORMANCE WITHIN 3 HOURS AFTER A MEAL

NO PERFORMANCE WITHIN 1/2 HOUR OF SEVERE EXERCISE (M092, M171, M131)

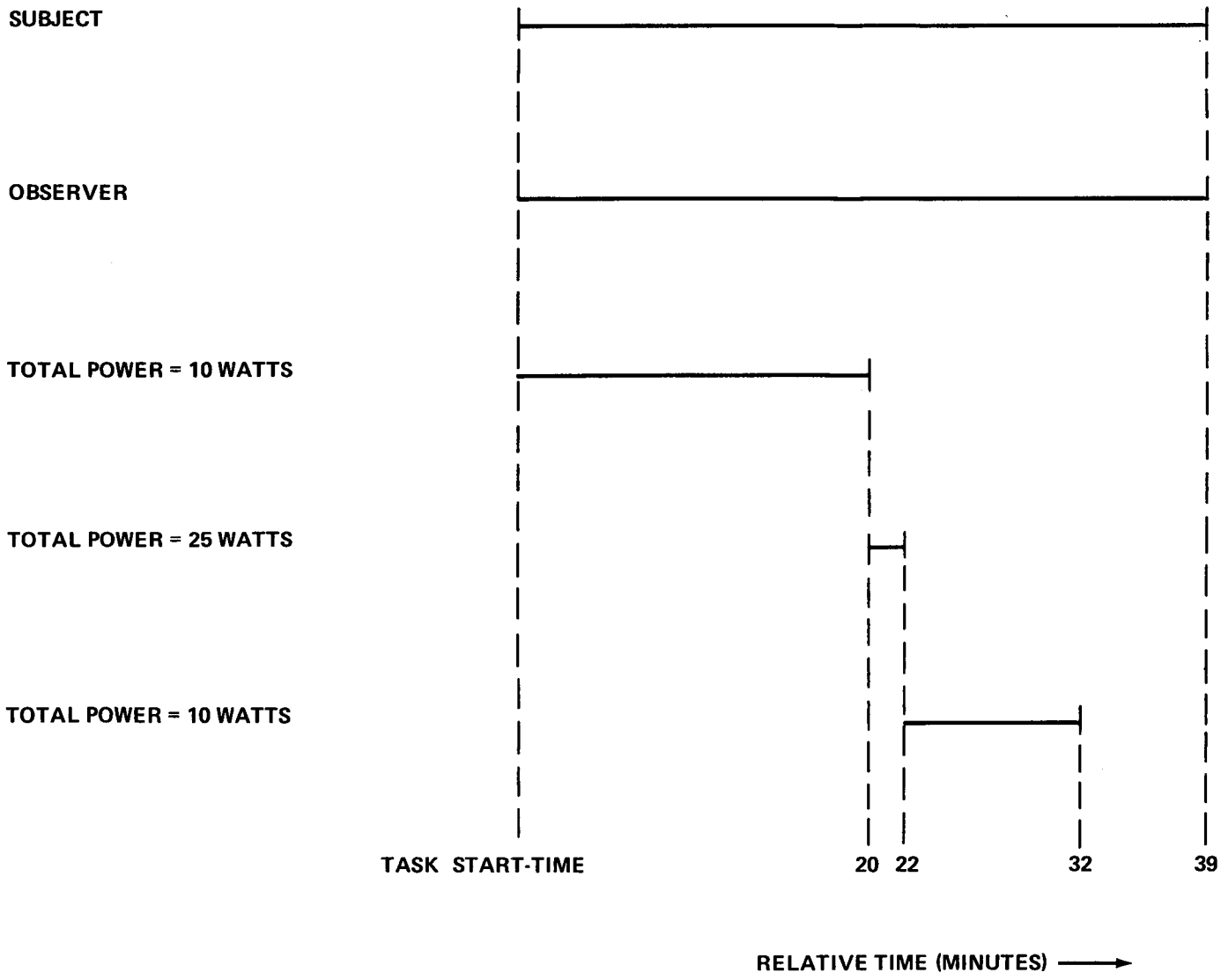


FIGURE 4.3 – SKYLAB EXPERIMENT M093: VECTORCARDIOGRAM TIMELINE DIAGRAM FOR ONE PERFORMANCE ON ONE CREWMAN.

Figure 4.4

Task Description Cards for M093A

```
M093A/TITLE,   1                    TASK M093 - VECTORCARDIOGRAM
M093A/COMMNT,  1     M093A DENOTES SUBJECT IS CREWMAN A.
M093A/COMMNT,  2     TASKS M131 AND M171 EACH HAVE 3 MODES.  THE
M093A/COMMNT,  3     FIRST LETTER FOLLOWING THE BASIC TASK NAME DENOTES
M093A/COMMNT,  4     THE MODE.  THE SECOND LETTER DENOTES THE
M093A/COMMNT,  5     PRINCIPAL SUBJECT.
M093A/PRI,     20
M093A/OBJEC,   9, 9,      03:00:00,     00:08:00
M093A/RES,     CREWA,     00:00:00,     00:00:39
M093A/INHIB,   BREAK,     00:01:30,     00:04:30
M093A/INHIB,   DINNER,    00:01:15,     00:04:15
M093A/INHIB,   LUNCHA,    00:01:15,     00:04:15
M093A/INHIB,   M131AA,    00:00:55,     00:01:55
M093A/INHIB,   M131BA,    00:00:40,     00:01:40
M093A/INHIB,   M131CA,    00:00:40,     00:01:40
M093A/INHIB,   M092A,     00:01:16,     00:02:16
M093A/INHIB,   M171AA,    00:01:30,     00:02:30
M093A/INHIB,   M171BA,    00:01:25,     00:02:25
M093A/INHIB,   M171CA,    00:02:00,     00:03:00
M093A/RES,     POWER1,    00:00:00,     00:00:20,     10
M093A/RES,     POWER2,    00:00:20,     00:00:22,     25
M093A/RES,     POWER3,    00:00:22,     00:00:32,     10
M093A/RES,     ANY,       00:00:00,     00:00:39
```

As indicated on the Comment Cards, every version of a task must be defined as a separate task. Thus, M093 becomes three tasks: M093A, M093B, and M093C which require crewmen A, B, and C respectively as the principal subject. The task descriptions for M093B and M093C would be virtually identical to M093A except that the third identifier field of the appropriate Resource and Inhibit Cards would be changed to reflect the proper subject.

In order to properly specify the operational restrictions on the task performance, the user must understand the meaning of the restriction and how it applies to each of the related tasks. For example, the restriction on the performance of M093A within three hours after the completion of a meal translates into inhibit restrictions on each of the subject's three meals: breakfast, lunch, and dinner. Since it was assumed that the crewmen would eat breakfast and dinner together, an inhibit restriction was specified for tasks BREAK and DINNER. It was further assumed that each crewman's lunch period would be scheduled separately. Since the performance restriction applies only to the subject's lunch period, the Inhibit Card specifies a restriction on task LUNCHA.

As discussed in Section 4.2.1.6, the data fields on each Inhibit Card contain the endpoints of the inhibited interval relative to the start-time of the independent task. Hence a knowledge of the time of the subject's final participation in the independent task relative to its start-time is required if the data fields on the Inhibit Card are to be specified correctly. For example, Figure 4.1b shows that crewman A's participation in the breakfast task ends at one hour and thirty minutes after the start of the task. Hence, the inhibited interval on the start-time of M093A relative to the start-time of the breakfast task begins at the end of this participation (00:01:30) and ends three hours later (00:04:30). All endpoints for enable and inhibit restrictions are similarly derived.

Finally, note that neither the M093 Timeline Diagram nor the M093 Task Description Cards make any reference to the three pieces of equipment specified in Figure 4.2. As noted in the figure, the equipment is only shared with tasks which cannot be performed at the same time as M093. Since there is no potential conflict, no Resource Cards for the equipment need be specified.

4.3.2 Use of an Equivalence Card

When an Equivalence Card is used to duplicate a task description, that card must be used to introduce the name of the new task. Any cards used to modify the duplicated description must appear after the Equivalence Card. The rules for modifying an existing task description are presented in Section 4.4.

4.4 Modifying a Task Description

The primary reason for establishing a data bank is to relieve the user of having to input all of the task descriptions to the Schedule Generator from punched cards every time the program is used. If a bank is to be established and used, the requirement to modify the task description data is twofold.

1. The capability is needed in the Data Bank Generator to edit the descriptions stored in the bank, thus enabling the user to easily incorporate permanent modifications to the task descriptions.

2. The capability is needed in the Scheduling Generator to permit the user to alter the task descriptions copied from the data bank. This capability enables the user to create unique task descriptions for the duration of the scheduling process without changing the descriptions stored in the data bank.

The TDL permits three types of edits: additions, deletions, and changes. Edits are performed card by card using the Descriptor and Annotation Cards described above. The first identifier field on the edit card contains the name of the task to be edited. The program scans all of the cards in the appropriate card group (Descriptor or Annotation) for that task until a card is found whose identifier fields exactly match the corresponding identifier fields on the edit card. When a match is found, the data fields on the stored card are deleted and replaced with the data fields on the edit card. If, however, the word DELETE appears in

the first data field, the entire card is deleted.* If no match is found, the edit card is added to the end of the appropriate card group.

This editing procedure requires that each card in a set of task description cards contain a unique set of identifiers. When more than one requirement is specified on the same resource in the same task description (Section 4.2.1.7.1), a different number must be affixed to the resource name in each requirement to maintain that uniqueness and thus permit the editing logic to differentiate between the two requirements.

4.5 The Description Card Data Deck

A particularly important feature of the Task Description Language is that it permits edit cards and description cards for new tasks to be interleaved when forming a data deck, so long as the relative position of the cards for each new task conforms to the input sequence rules defined in Section 4.3. This flexibility is achieved because each card is self-contained (i.e., it contains all of the information required to identify and modify a particular task description). The only exception occurs when more than one punched card is needed to make up a Descriptor Card. In that case, all of the punched cards making up the type card must be input together in the proper order.

A simple illustration of this flexibility is shown in Figure 4.5. Figures 4.1a and b show sets of description cards for the Sleep and Breakfast tasks in the order in which they would be input to the program. An alternative input sequence for each set of cards is shown in Figures 4.5a and b respectively. Though more difficult for the user to read, these alternatives are consistent with the four input sequence rules defined in Section 4.3. If these alternatives were used, the respective task descriptions would actually be stored in the sequence shown in Figure 4.1.

---

*Annotation Cards have no delimiter after the sequence number in the third identifier field. When the DELETE option is used, however, a comma should be placed in the column immediately following the sequence number and the word DELETE placed in the next field.

Figure 4.5a

Alternative Input Sequence for the Sleep Task

```
SLEEP/TITLE,   1                    TASK SLEEP
SLEEP/PRI,     1
SLEEP/COMMNT,  1          ALL CREWMEN SLEEP SIMULTANEOUSLY FOR ONE
SLEEP/COMMNT,  2          CONTINUOUS 8-HOUR PERIOD EVERY 24 HOURS.
SLEEP/OBJEC,   27, 27,        01:00:00,     00:00:00
SLEEP/TIME,    00:09:25
SLEEP/COMMNT,  3          THE FIRST PERIOD SHOULD NOT BEGIN PRIOR
SLEEP/RES,     CREWA,         00:00:00,     00:08:00
SLEEP/COMMNT,  4          TO 00:09:25 MISSION ELAPSED TIME.
SLEEP/RES,     CREWB,         00:00:00,     00:08:00
SLEEP/RES,     CREWC,         00:00:00,     00:08:00
```

Figure 4.5b

Alternative Input Sequence for the Breakfast Task

```
BREAK/TITLE,   1                    TASK BREAK
BREAK/COMMNT,  1       CREWMEN EAT BREAKFAST TOGETHER IMMEDIATELY AFTER
BREAK/PRI,     2
BREAK/COMMNT,  2       AWAKENING FROM SLEEP.  ONE HOUR AND 30 MINUTES
BREAK/OBJEC,   27, 27,        01:00:00,     00:00:00
BREAK/ENABLE,  SLEEP,         00:08:00,     00:08:00
BREAK/RES,     CREWA,         00:00:00,     00:01:30
BREAK/COMMNT,  3       ARE ALLOTTED FOR BREAKFAST.
BREAK/RES,     CREWB,         00:00:00,     00:01:30
BREAK/RES,     CREWC,         00:00:00,     00:01:30
```

More practical use of this flexibility is shown in Figures 4.6 through 4.8. Figure 4.6 shows a data deck that would be used to edit the three task descriptions shown in Figures 4.1 and 4.4. The first two cards in Figure 4.6 edit the original set of description cards for Task Sleep (Figure 4.1a) by substituting the two cards in the edit deck for the corresponding cards having the same identifier fields. The set of cards describing the Task Breakfast (Figure 4.1b) is similarly edited by substituting the Priority Card in the edit deck for the one stored in the original task description. Finally, the three Comment Cards for Task M093A are added to the set of description cards for that task since there are no corresponding cards in the original set (Figure 4.4) that have the same identifier fields. The task descriptions resulting from these edits are shown in Figures 4.7 and 4.8.

Figure 4.6

Data Deck to Edit Tasks Sleep, Breakfast, and M093A

```
SLEEP/COMMNT,    4          TO 00:10:00 MISSION ELAPSED TIME
SLEEP/TIME,      00:10:00
BREAK/PRI,       4
M093A/COMMNT,    6      THE 8-HOUR TOLERANCE SPECIFIED ON THE OBJECTIVE
M093A/COMMNT,    7      CARD IS A WORKING VALUE AND NOT A DEFINITE
M093A/COMMNT,    8      REQUIREMENT.
LAST
```

Figure 4.7a

Task Description for Task SLEEP After Editing

```
SLEEP/TITLE,     1                    TASK SLEEP
SLEEP/COMMNT,    1          ALL CREWMEN SLEEP SIMULTANEOUSLY FOR ONE
SLEEP/COMMNT,    2          CONTINUOUS 8-HOUR PERIOD EVERY 24 HOURS.
SLEEP/COMMNT,    3          THE FIRST PERIOD SHOULD NOT BEGIN PRIOR
SLEEP/COMMNT,    4          TO 00:10:00 MISSION ELAPSED TIME.
SLEEP/PRI,       1
SLEEP/OBJEC,     27, 27,     01:00:00,     00:00:00
SLEEP/TIME,      00:10:00
SLEEP/RES,       CREWA,      00:00:00,     00:08:00
SLEEP/RES,       CREWB,      00:00:00,     00:08:00
SLEEP/RES,       CREWC,      00:00:00,     00:08:00
```

Figure 4.7b

Task Description for Task BREAK After Editing

```
BREAK/TITLE,     1                    TASK BREAK
BREAK/COMMNT,    1     CREWMEN EAT BREAKFAST TOGETHER IMMEDIATELY AFTER
BREAK/COMMNT,    2     AWAKENING FROM SLEEP.  ONE HOUR AND 30 MINUTES
BREAK/COMMNT,    3     ARE ALLOTTED FOR BREAKFAST.
BREAK/PRI,       4
BREAK/OBJEC,     27, 27,     01:00:00,     00:00:00
BREAK/ENABLE,    SLEEP,      00:08:00,     00:08:00
BREAK/RES,       CREWA,      00:00:00,     00:01:30
BREAK/RES,       CREWB,      00:00:00,     00:01:30
BREAK/RES,       CREWC,      00:00:00,     00:01:30
```

Figure 4.8

Task Description for TASK M093A after Editing

```
M093A/TITLE,     1                    M093 - VECTORCARDIOGRAM
M093A/COMMNT,    1    M093A DENOTES SUBJECT IS CREWMAN A.
M093A/COMMNT,    2    TASK M131 AND M171 EACH HAVE 3 MODES.  THE
M093A/COMMNT,    3    FIRST LETTER FOLLOWING THE BASIC TASK NAME DENOTES
M093A/COMMNT,    4    THE MODE.  THE SECOND LETTER DENOTES THE
M093A/COMMNT,    5    PRINCIPAL SUBJECT.
M093A/COMMNT,    6    THE 8-HOUR TOLERANCE SPECIFIED ON THE OBJECTIVE
M093A/COMMNT,    7    CARD IS A WORKING VALUE AND NOT A DEFINITE
M093A/COMMNT,    8    REQUIREMENT.
M093A/PRI,       20
M093A/OBJEC,     9, 9,        03:00:00,     00:08:00
M093A/RES,       CREWA,       00:00:00,     00:00:39
M093A/INHIB,     BREAK,       00:01:30,     00:04:30
M093A/INHIB,     DINNER,      00:01:15,     00:04:15
M093A/INHIB,     LUNCHA,      00:01:15,     00:04:15
M093A/INHIB,     M131AA,      00:00:55,     00:01:55
M093A/INHIB,     M131BA,      00:00:40,     00:01:40
M093A/INHIB,     M131CA,      00:00:40,     00:01:40
M093A/INHIB,     M092A,       00:01:16,     00:02:16
M093A/INHIB,     M171AA,      00:01:30,     00:02:30
M093A/INHIB,     M171BA,      00:01:25,     00:02:25
M093A/INHIB,     M171CA,      00:02:00,     00:03:00
M093A/RES,       POWER1,      00:00:00,     00:00:20,     10.0
M093A/RES,       POWER2,      00:00:20,     00:00:22,     25.0
M093A/RES,       POWER3,      00:00:22,     00:00:32,     10.0
M093A/RES,       ANY,         00:00:00,     00:00:39
```

5.0 The ATS Internal Data Structure

Ordinarily, A.S.A.* FORTRAN permits only static storage allocation: i.e., core storage can only be allocated prior to the program's execution. When writing a program therefore, the programmer must estimate the maximum amount of data expected for each data variable and then allocate sufficient core storage to accommodate that maximum. Once the program is in execution, the space allocated to a particular variable (the data array) cannot be used by any other variable even if part or all of that assigned space is not needed. However, during the execution of the Schedule Generator there will be wide variations in the quantity of data generated in each of the three major data types (task descriptions, commitment tables, and start-time windows). Hence, the A.S.A. FORTRAN system of static storage allocation will not provide an efficient use of the available core storage.

A more efficient use of core storage can be achieved with dynamic storage allocation. In this method, the dimension of a larger linear array (called a Working Array) is defined prior to the program's execution. Then, during execution, core space within the Working Array can be allocated to different variables as needed. There is no limit to the number of different variables to which space can be allotted or the amount allotted to each variable. The only restriction is that the sum total of all core space allotted to all variables be less than (or equal to) the fixed size of the Working Array.

Two methods of dynamic allocation were used in the ATS: dynamic array storage and linked-list storage. Dynamic array storage was used to store the resource and ephemeris commitment tables while linked-list storage was used to store task descriptions and start-time windows. To further decrease the amount of required core storage, auxiliary drum storage was used to store all of the task descriptions. The descriptions are copied into core storage from the drum when needed. Since the Schedule Generator considers only one task at a time, no more than one task description is ever in core storage at any particular time.

5.1 Dynamic Array Storage

5.1.1 Characteristics of Array Storage

In dynamic array storage, the Working Array is partitioned off into smaller areas, with each area containing a different set of data. Within each area, data is stored in

---

*American Standards Association

consecutive locations (the ith data item is stored in the ith location) and so a particular item can only be accessed by knowing its relative position within the particular data set (and hence within the large Working Array). A small linear array is therefore used as a "table of contents" to the Working Array. The small array contains the name of each data set, its starting location in the Working Array, and the number of data items in the set. Any data item can therefore by accessed by searching over the portion of the Working Array defined by the starting location and number of items for the pertinent data set. Dynamic allocation of core storage is achieved by permitting the number and size (number of core locations) of the partitioned areas to vary as long as the total number of locations required does not exceed the dimension of the Working Array.

Three advantageous features of array storage are

1. It permits rapid access time to specific data items.

2. It facilitates searches of the data set when the data elements are monotonically increasing.

3. It maximizes the efficiency of data storage by requiring only one core location per data item.

The second of these advantages is particularly important. During the generation of a schedule, the commitment tables will be continually searched to find periods of resource availability. Since the number of table searches made during the generation of a schedule will be quite high, a significant reduction in running time results from storing the commitment tables in an array configuration.*

A disadvantage of array storage is the degree of difficulty with which a data item can be inserted into a data

---

*The computation time required for each search is further reduced by performing a binary rather than a sequential search of the data in the table. As shown in Reference 4, a binary search always requires n comparisons when the number of table entries lies between 2^{n-1} and 2^{n}. On the average, the number of comparisons for a corresponding sequential search equals one half the number of table entries. Hence, the saving becomes quite significant for large tables.

set. If an item is to be inserted at the ith location in a set containing n items, the existing items in the ith through the nth locations must each be shifted to the next higher location so that the data item may be inserted without destroying any of the existing data. (The reverse process is required to delete an item.) As the number of items in a data set increases, the amount of data that must be shifted for each insertion also increases and that amount soon becomes quite significant and time consuming.

5.1.2 The Use of Dynamic Array Storage in the Schedule Generator

In the Schedule Generator, a large one-dimension array, the WA Array,* has been set aside to store all of the resource tables. At the beginning of each run, all of the ephemeris resource tables on the ATS Ephemeris Tape are read directly into the array from magnetic tape.** Portions of the remaining area in the array are allocated to each of the resource commitment tables, the size of each allotment being determined by the user. For each resource (except ephemeris resources), the user must specify the name, type (binary or analog), and the maximum number of entries permitted. The allotted size is then the product of the maximum number of entries and the number of columns required for the particular table.

Figure 5.1 shows a portion of the WA Array at some intermediate point in the scheduling process. A small linear array named LTABLE is used as a table of contents to the WA Array. Each entry contains the name of the array, the type of array, the maximum number of entries permitted, the starting location, and the current number of entries.

The process of making a new entry into a table is illustrated in Figure 5.2. A search of the binary resource table has determined that a new entry is to be inserted between the first and second entries in the table (the three arrows labeled A in Figure 5.1). The original data is sequentially shifted within the allocated area to vacate the required locations and the three data items comprising the new entry are inserted in the vacated locations. The process may be repeated until all of the locations within the allocated area are occupied. Any attempt to insert more than the maximum allowable entries in any resource table will terminate the execution of the program with a diagnostic message (Appendix A - Section A.2.2).

---

*The present size of the WA Array is 12,000 locations.

**Input of ephemeris information is optional.

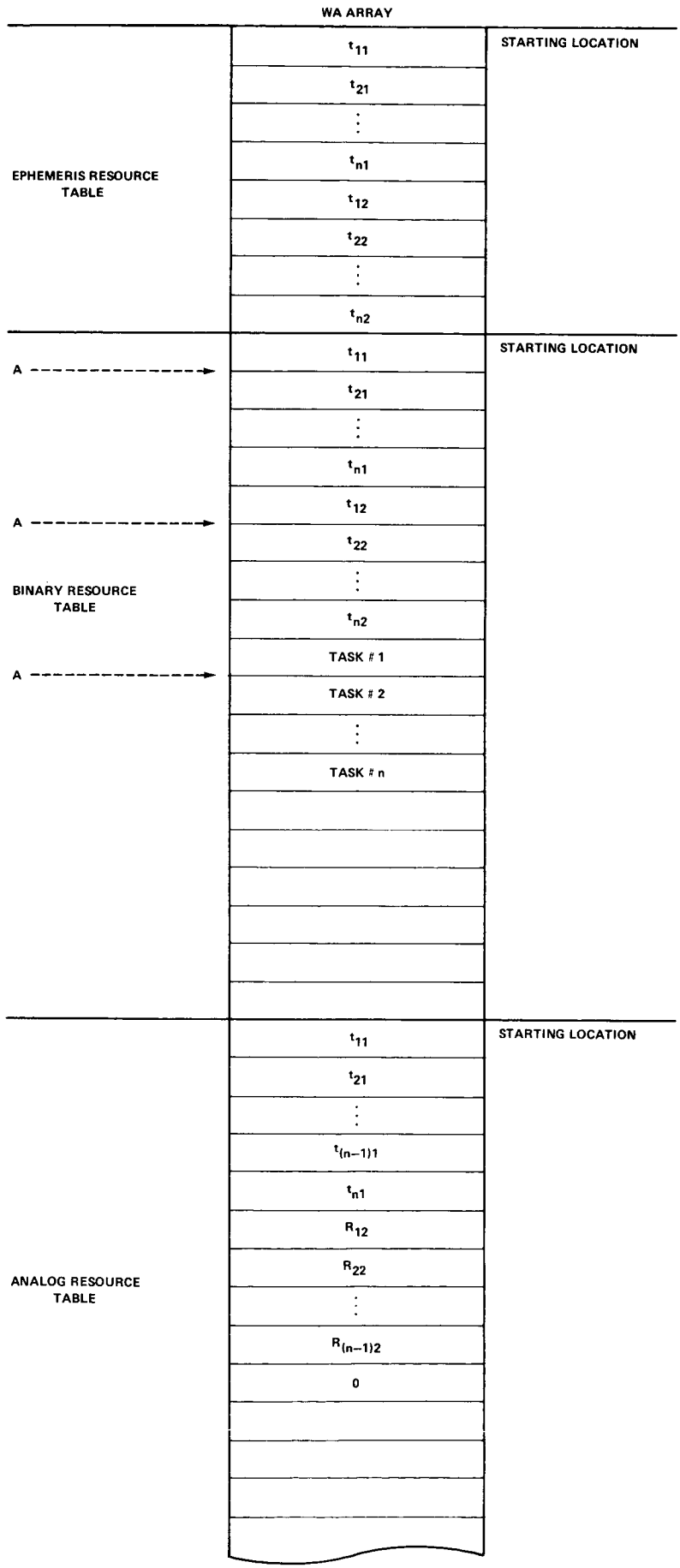


FIGURE 5.1 – DYNAMIC STORAGE OF RESOURCE TABLES

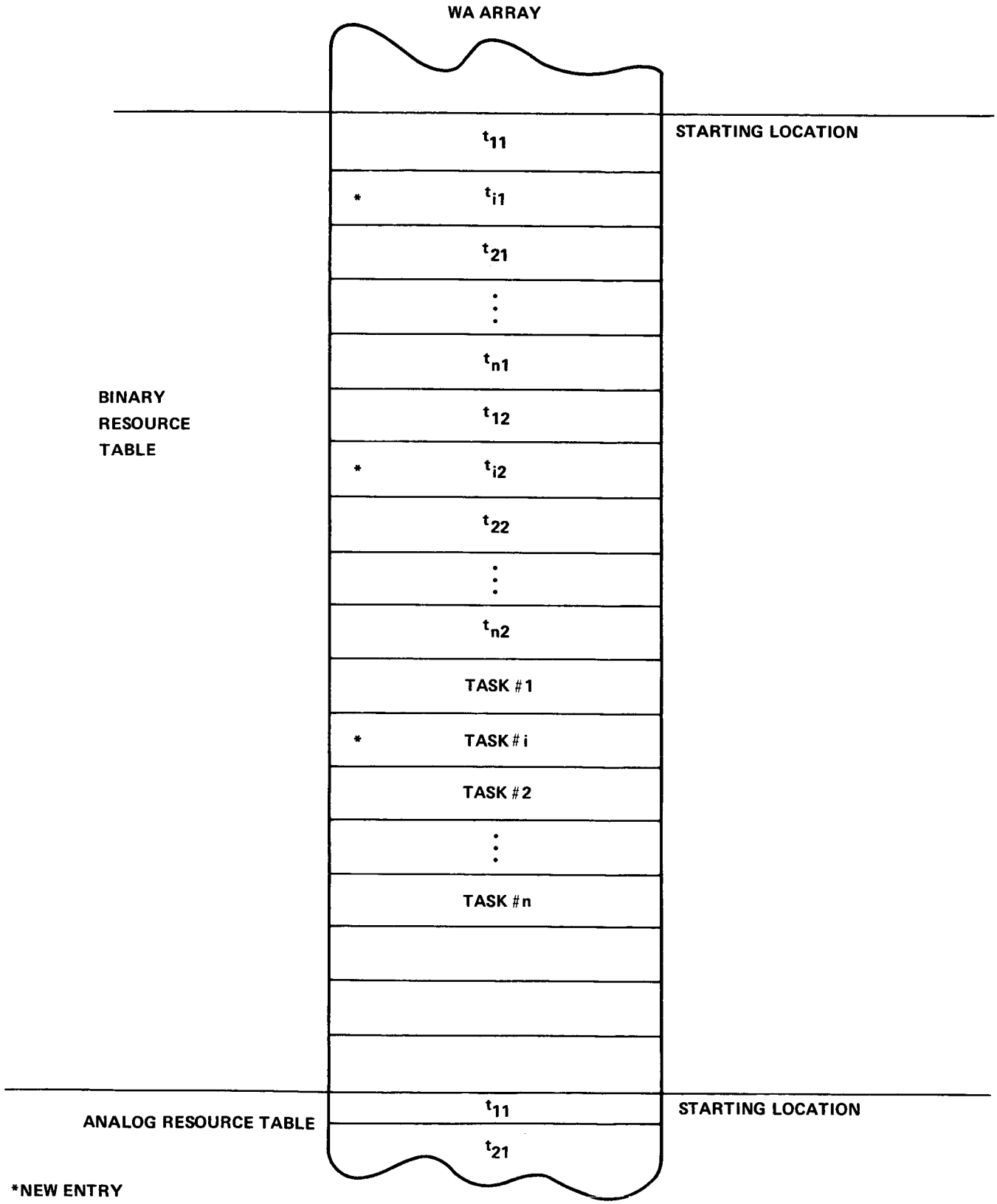


FIGURE 5.2 – INSERTING DATA INTO A RESOURCE COMMITMENT TABLE.

5.2 Linked-List Storage

5.2.1 Characteristics of Linked-List Storage

In linked-list storage, each data item is stored together with the core address of the next logical data item. The core address is called a pointer and the combination of data item and core address is called a data node or link (illustrated in Figure 5.3a). The nodes are linked together to form a list of data items as shown in Figure 5.3b. Since each link in the list contains a pointer to the following link, data items can be accessed by following the pointers rather than by depending upon a knowledge of the item's relative position in an array. Hence, the need for physically sequential storage is eliminated.

The list structure shown in Figure 5.3b is called an elementary list because each link contains a single data item (DI) in its element field. More complex lists can be constructed by replacing any of the data items in the elementary list with the address of another list which is considered a sublist of the primary list. The basic structure is illustrated in Figure 5.3c. Very complex tree-like branching lists can be accommodated by repeated use of this simple feature.

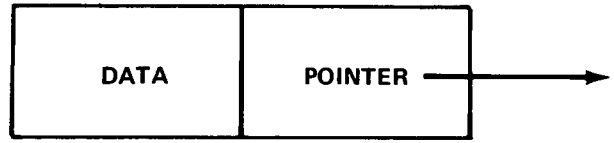
When list structures are used, one of the first steps in the program's execution is to form a single list of empty data nodes (usually called the Available Space List or ASL) from the designated Working Array. After the ASL is formed, data lists are created or lengthened by removing data nodes as needed from the ASL. When a data item is no longer needed, the node containing that item is returned to the ASL, thus achieving dynamic storage allocation. (The only limitation to the allocation of data nodes is that the total number of cells being used at any particular moment cannot exceed the total number of cells originally created.)

The primary advantages of linked-list storage include:

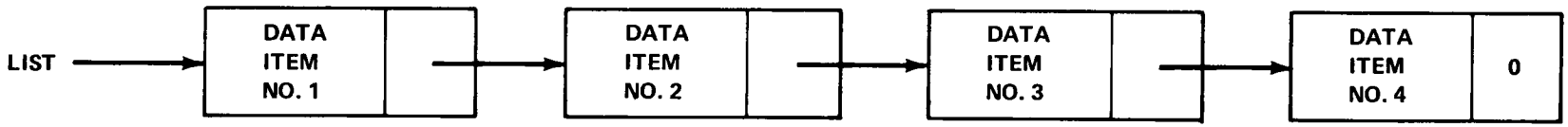
1. That wide variations in the type and quantity of data items can be easily accommodated.

2. That data items in a list structure can be rearranged (e.g., added, deleted, modified, or shifted in position) easily by changing the appropriate pointers.

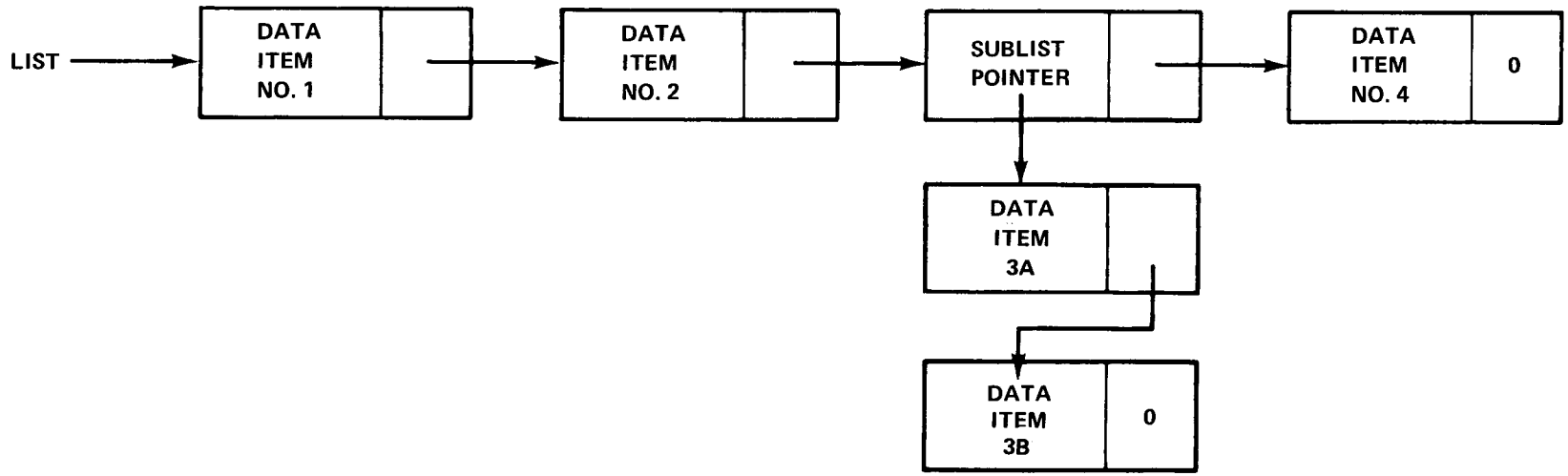
3. That it facilitates the formation of more intricate data structures by permitting the formation of complex lists.



a) DATA NODE CONFIGURATION



b) ELEMENTARY LIST STRUCTURE



c) COMPLEX LIST STRUCTURE

FIGURE 5.3 - LINKED LIST STRUCTURE

Linked-list storage does have two distinct disadvantages. As noted above, individual data items can only be accessed by following the pointers contained in each data node; hence, average access time is significantly higher compared to array storage. In addition, at least one core location at each data node must be allocated for information related to the data structure (e.g., the pointer address). In effect, this allocation imposes an additional storage penalty (beyond the required single core location) on each data item stored. The use of list structures therefore is usually limited to applications where the amount of data is unpredictable and the number of accesses will be relatively few (e.g., start-time windows and task descriptions).*

5.2.2 The SAC-1 List Processing System

The SAC-1 (system for Symbolic and Algebraic Calculations - Version 1) list-processing language** was selected for use in the ATS because, unlike most other list-processing languages, it is compatible with A.S.A. FORTRAN. This compatibility exists because the entire SAC-1 system, with the exception of a few "primitive" subprograms, is written in A.S.A. FORTRAN. The primitives are machine dependent and are usually written in the assembly language designed for the particular computer.

5.2.2.1 The SAC-1 List Structure

Each link or cell in the SAC-1 system consists of two sequential core locations or words. A cell is divided into four fields: The first three fields are contained in the first word while the element field completely occupies the second word. The cell structure is illustrated in Figure 5.4. The successor field always contains the pointer to the next cell on the list while the element field may contain either a data item or a pointer to another cell. (The pointer is stored as the actual core address of the first word of the referenced cell.) The type field contains the value of one or zero depending upon whether the content of the element field is an address or a data item. Finally, the reference count field contains an integer indicating the number of different successor fields which are currently pointing to the particular cell. The type and successor fields are required to permit the formation of more complex list structures. A complex list using the SAC-1 cell structure is shown in Figure 5.5.

---

*A more complete discussion of list structures can be found in References 5 and 6.

**References 7 and 8.

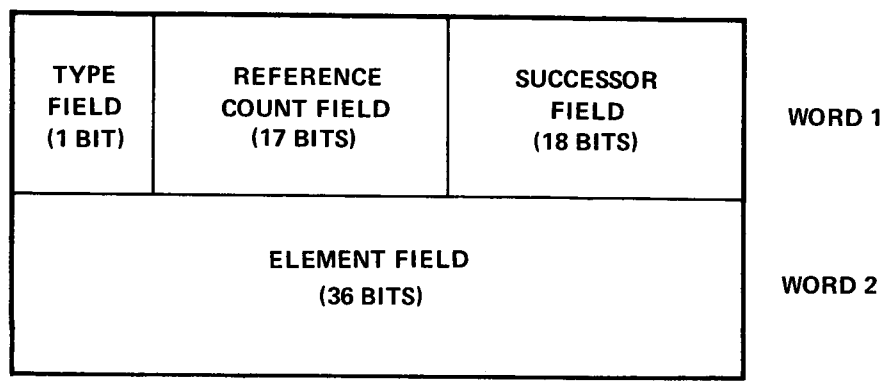


FIGURE 5.4 – SAC-1 CELL STRUCTURE.

02

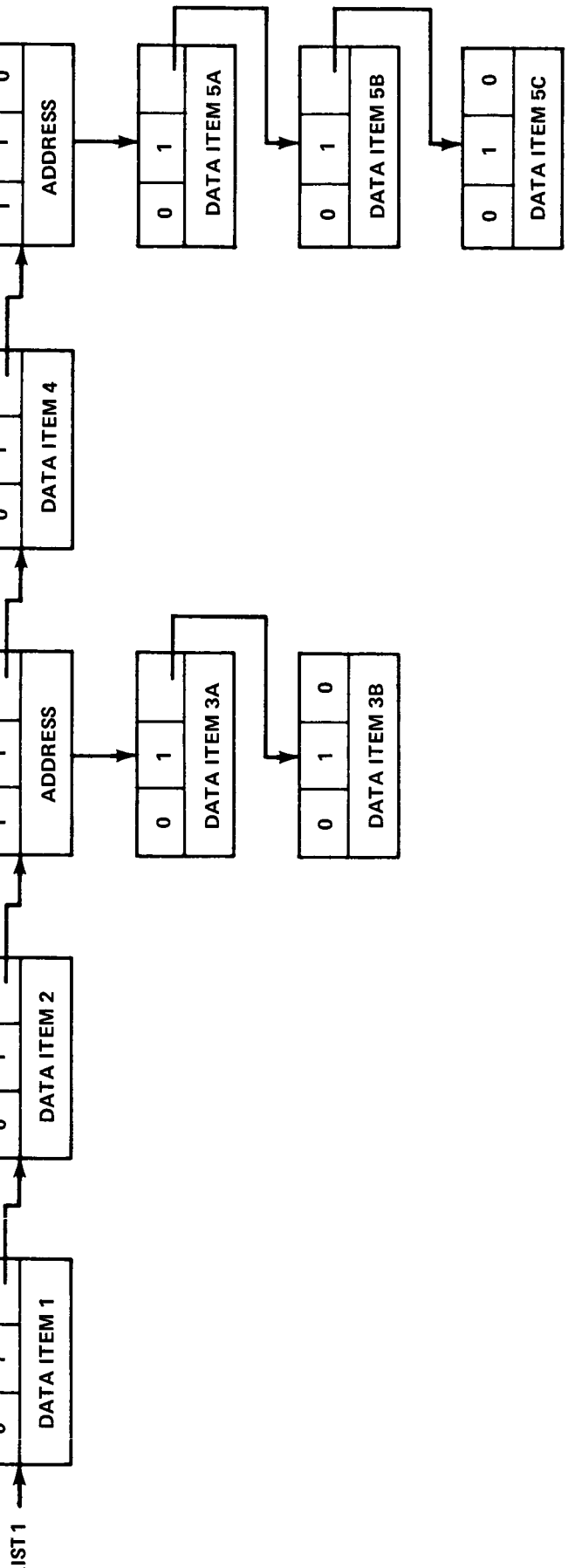


FIGURE 5.5 – LIST STRUCTURE IN SAC-1.

5.2.2.2 SAC-1 Operations

A different subroutine is used to store information into each of the four cell fields and, correspondingly, a different function subprogram is used to retrieve data from each of the cell fields. The subroutines and function subprograms used to access each of the cell fields are listed in Table 5.1. As the table shows, the FORTRAN statement

```
CALL ALTER(X,P)
```

will place the data stored in location X into the element field of the cell P. Similarly the FORTRAN statement

```
X=FIRST(P)
```

will retrieve (non-destructively) the contents of the element field of the cell P and place it in location X. In these statements X is the name of a FORTRAN variable into (or from) which the contents of the element field are to be transferred. The entire word is transferred as is so that the bits may represent a floating point number, an integer, or a string of Hollerith characters.* P represents the name of a FORTRAN integer variable that contains the core address of the desired cell. The address is always stored as an integer.

Similarly, the FORTRAN variables R, S, and T in Table 5.1 contain data of the same form as is in the reference count, successor, and type fields, respectively. They must be stored in integer form. Note that the contents of T are restricted to the values zero and 1, since the type field occupies only 1 bit.

As shown above, a particular data item can be accessed only by specifying the core address of the cell containing that data item, and so a FORTRAN variable must be defined which contains the address of the cell. If, for example, the FORTRAN variable LIST1 (Figure 5.5) contains the address of cell 1, the first data item on the list can be retrieved by using the Function FIRST. Thus, as a result of the FORTRAN statement

```
DI1=FIRST(LIST1)
```

---

*Since the bits are transferred intact, it becomes the user's responsibility to keep track of the type of data in each element field so that during retrieval, the data is transferred to an appropriately named (integer or real) location.

Table 5.1

Subroutines and Function Subprograms

Used to Access SAC-1 Cell Fields

	Store*	Retrieve**
Element Field	CALL ALTER(X,P)	X=FIRST(P)
Reference Count Field	CALL SCOUNT(R,P)	R=COUNT(P)
Successor Field	CALL SSUC(S,P)	S=TAIL(P)
Type Field	CALL STYPE(T,P)	T=TYPE(P)

*The data stored in variable X, R, S, or T is placed in the appropriate field of the cell P.

**Data is retrieved from the appropriate field of cell P and placed in variable X, R, S, or T.

FORTRAN variable DI1 contains data item 1. To access any other data item on the list, the pointer on successive cells must be followed until the location of the desired data item is found. Thus, to access the data item in cell 2, we define the FORTRAN variable POINT=TAIL(LIST1). The variable POINT will contain the contents of the successor field in cell 1 which is, by definition, the location of cell 2. Then data item 2 can be retrieved by defining the FORTRAN variable DI2=FIRST(POINT). In a similar fashion, the data item in cell 3A can be retrieved by successive calls to Functions FIRST and TAIL. Thus

POINT=TAIL(LIST1)	POINT contains the address of cell 2
POINT=TAIL(POINT)	POINT contains the address of cell 3
POINT=FIRST(POINT)	POINT contains the address of cell 3A
DI3A=FIRST(POINT)	DI3A contains data item 3A

Note that these operations have no effect on the original list structure defined by the variable LIST1.

The accessing of the four cell fields are the lowest level functions in the SAC-1 system and the eight subprograms used to perform these operations are called the SAC-1 primitives. Higher level routines, using these primitives, perform other list-processing functions: creating new lists, adding items to existing lists, and erasing lists (i.e., returning all cells on the list to the ASL).

5.2.3 Task Description Lists

All task descriptions in the ATS are stored in a linked list structure. Each task description is stored as two separate lists; one contains the alphanumeric information on the Annotation Cards and the other contains the data on the Descriptor Cards. The relationship between the input cards and the corresponding list structure can be illustrated by comparing the Descriptor Cards for Task Sleep in Figure 4.1a and the internal list structure resulting from those cards shown in Figure 5.6. Each card, with the exception of the Priority Card, corresponds to a separate sublist. The contents of each field on a card is stored on a separate cell. Identifier fields as well as resource names are stored in their Hollerith representation while all values of time are stored in minutes, the common unit of time for the ATS system.

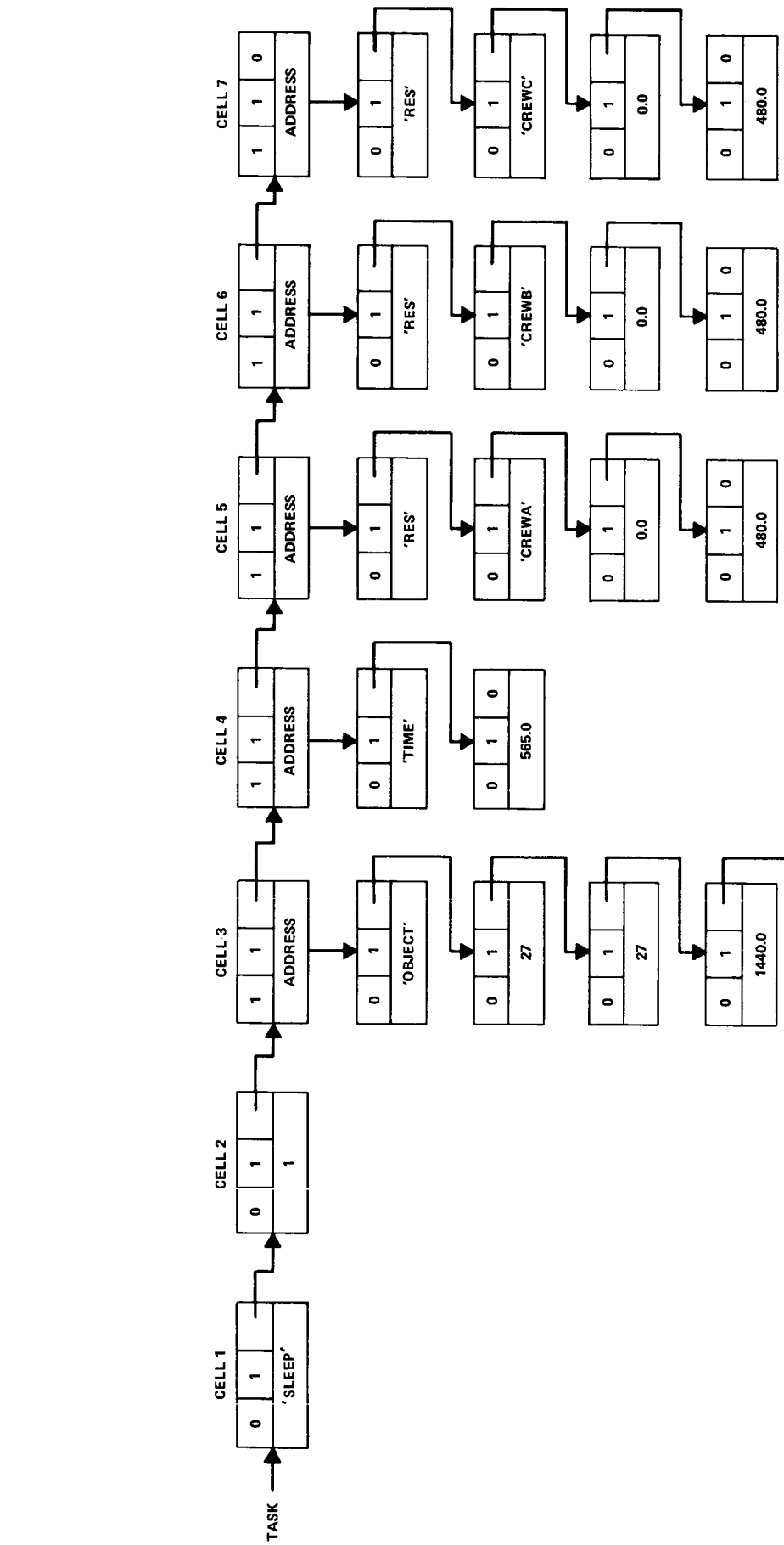


FIGURE 5.6 – LIST STRUCTURE FOR TASK SLEEP.

The input sequence rules in Section 4.3 require that the Priority and Objective Cards be the first two Descriptor Cards input for any task. The Priority Card is used to create a new Descriptor List with the task name entered in cell 1 and the task priority entered in cell 2. Thereafter, any card containing the same name in its first identifier field directs the program to that particular list. The information on the Objective Card is always formed as a sublist to cell 3.

The list structure corresponding to a set of Annotation Cards is almost identical to the structure shown in Figure 5.6. The Annotation List also has the task name in cell 1 but omits a second data cell. Thereafter each sublist contains 13 cells; the first cell contains the card identifier and the remaining 12 contain the alphanumeric information exactly as it appears on the card. The information is stored in six-character (1 word) blocks, one to each cell. The second and third identifier fields on the Annotation Cards are combined by the program into one identifier which is stored in the first cell of each sublist. The first character of the identifier is the letter 'T' or 'C' depending upon the card type (Title or Comment, respectively). The remaining characters comprise the card number from the third identifier field.

5.2.4 Other Uses of the SAC-1 System

The availability of the SAC-1 system led to the use of list structures for other applications. In the Schedule Generator, the endpoints of the start-time windows as well as the start-times of each task are stored as lists. These applications will be explained as part of the functional descriptions of the Schedule Generator (Section 7.0).

5.3 Auxiliary Storage

5.3.1 Storage Characteristics

The Bellcomm time-shared computer system gives low priority and poorer service to programs with large core requirements but does not penalize programs that use the auxiliary drum storage facilities. It is therefore advantageous to make use of these auxiliary facilities wherever feasible.

In the ATS, the bulk and accessing requirements of the task description data make it highly desirable to store these descriptions on drum storage. The ATS can accommodate up to 200 tasks with each containing a minimum of four (and usually many more) task description cards. If the list structures for all of the task descriptions were in core simultaneously,

the number of SAC-1 cells required would be prohibitively high. The Scheduling Generator however refers to only one task Descriptor List at a time, since the tasks are processed sequentially. These factors led to a decision to store the bulk of the task description data on auxiliary storage, bringing into core only one set at a time, as needed.

There are four distinct types of auxiliary bulk storage available on the Bellcomm computer system -- magnetic tape, two classes of magnetic drum, and FASTRAND (which is also a drum, but of enormous capacity). Tape is not a suitable medium for storing the task descriptions because it allows only sequential reading and writing operations, whereas the task data will have to be retrieved in a data dependent order. (Tape is usually used to store the Permanent Data Bank, the ephemeris resource tables, and the History Tape, although they can equally well be kept on FASTRAND.) The characteristics of the three drum devices are shown in Table 5.2. Logically, all three devices appear identical to the user. The unit to be used can be specified on a control card at run time, or alternatively may be left to the system to assign based on availability. Ordinarily, the two fast drums are only used for scratch storage during execution of a program while FASTRAND is used for long term storage of programs and data.

Records may be written into and retrieved from an assigned file either sequentially or, with a little more effort, in arbitrary order. The latter option is used in ATS for the task descriptions. Each task description occupies its own standard sized record; the record size being large enough to accommodate a rather lengthy description. A special subroutine handles the retrieval and storage of each record via a similarly sized buffer block in core.

5.3.2 Storage of the Task Descriptions

5.3.2.1 The Symbolic Representation of a Linked-List

When a task description is stored in core in linked-list structure, its complex branching structure is defined through the use of the branching nodes (the cells containing sublist pointers in their element fields) as illustrated in Figure 5.6. However, a substitute notation is required to store the task description data in a linear array on the storage drum.

A linked-list structure may be represented symbolically by separating successive data items on a list by commas and enclosing the set of data items with parentheses. Using this notation, the simple list structure in Figure 5.3b would be represented as

Table 5.2

Characteristics of UNIVAC 1108 Auxiliary Drum Storage

Unit	Storage Capacity (Words)	Average Access Time (milliseconds)	Word Transfer Rate (words/second)
FH-432	262,144	4.3	240,000
FH-1782	2,097,152	17	240,000
FASTRAND II	22,020,096	92	25,590

(DI1,DI2,DI3,DI4)

Similarly, the symbolic representation of the list structure in Figure 5.5 would be

(DI1,DI2,(DI3A,DI3B),DI4,(DI5A,DI5B,DI5C))

5.3.2.2 Conversion of a Linked-List Structure to an Interval Representation of its Symbolic Form

In the ATS, task description data is removed from the linked-list structure and converted to a continuous string of Hollerith symbols of the form shown above. Six-bit codes for the right parenthesis, the left parenthesis, and the comma are inserted into the string where appropriate. The 36-bit data items are extracted from the element fields and included at the appropriate points in the string. The six-bit code representing the letter O is inserted into the string immediately preceding each data item. The six-bit code is used to signify the presence of a data item. When the code is detected during the reconstruction of a list, the next 36-bits are placed in the element field of an available cell without regard to the type of data (integer, floating point, or Hollerith) that it may represent.

Two subroutines are used to make these conversions. A call to Subroutine TWRITE

CALL TWRITE(O,X,BUFFER)*

converts List X to its symbolic form by extracting the data item from the element field of each cell on the list and inserting the necessary delimiters (commas and parentheses) where necessary. The resulting bit stream is placed in successive locations in Array BUFFER. Similarly, Function Subprogram TREAD is used to reconstruct the list from its symbolic representation. The statement

X=TREAD(O,BUFFER)*

will reconstruct the list described by the bit stream in Array BUFFER by placing each of the data items into the element field of an available data cell and connecting the cells in the manner indicated by the position of the delimiters.

---

*Reference 8.

5.3.2.3 Task Description Storage Files

As noted above, the task descriptions are stored on magnetic drum files for the duration of the run. The descriptions are stored on two separate files: the first file contains the symbolic representation of the Annotation List for each task while the second file contains the symbolic representation of the Descriptor List for each task. On each file, the data pertaining to each task is stored in a separate logical record. The records appear in the same sequence on both files. Therefore, when the ith record on the first file contains the Annotation List for a particular task, the ith record on the second file will contain the Descriptor List for the same task.

Each file is created by first constructing the symbolic representation of the linked-list in Array BUFFER (Section 5.3.2.1) and then writing the contents of the array onto the file. Note that the entire contents of Array BUFFER are written onto the file regardless of how many locations within the array were actually used for the symbolic representation of the list. The logical records are therefore all of equal length which permits them to be randomly accessed by specifying the record number.

The FORTRAN statement

```
CALL DWRITE (IUNIT, BUFFER, NWDS,K)*
```

writes a record of length NWDS from Array BUFFER onto the kth record of the drum file assigned as logical unit IUNIT. Subroutine DWRITE computes the relative address of the file's jth sector by assuming equal length records of size NWDS and equal length sectors of 28 words. Hence the relative address of the beginning of the kth record on the file becomes K(NWDS)/28. Note therefore that the size of Array BUFFER, variable NWDS, must be a multiple of 28.

Retrieving a record from the file is similarly performed by Subroutine DREAD where the statement

```
CALL DREAD (IUNIT, BUFFER, NWDS, K)*
```

causes the kth record of length NWDS to be copied from the file into Array BUFFER.

---

*Reference 8.

5.3.2.3 Summary

In summary, writing a Task Descriptor or Annotation List onto a drum file is a two-step process: first the conversion of the internal linked-list structure to a bit stream representing the symbolic representation of the list and second, the placement of that representation onto the drum file. Retrieving a list from the file is the logical inverse of the two-step writing process: the record containing the symbolic representation of desired list is copied from the appropriate file into Array BUFFER and then the linked-list structure is reconstructed from the contents of the array. Note that the dimension of Array BUFFER places a practical limit on the number of Descriptor and Annotation Cards that can be included in a single task description. In the current version of the ATS, Array BUFFER is dimensioned to 308 locations which means that a task description may have no more than 35 Descriptor Cards and no more than 17 Annotation Cards.

6.0 Functional Description of the Data Bank Generator

The Data Bank Generator is used to create and maintain a permanent Data Bank. The bank contains the descriptions of each task as well as a table of contents that indexes all of the tasks in the bank. The bank may be stored on magnetic tape or FASTRAND; its format and use are independent of the storage medium.

The format of the Data Bank is shown in Figure 6.1 where each rectangle represents a separate logical record. The first record on the file contains the value of the FORTRAN integer NTOC, the number of tasks stored in the bank. The TOC array in the second record serves as the bank's table of contents. The first column of each entry in the array contains the name of the task, the second column contains the task priority, and the third column contains the date of the task's last revision.

The remainder of the bank contains the task descriptions. Each description is stored on two sequential records. The first record contains the symbolic string representation of the Annotation Card data while the second record contains the Descriptor Card data stored in the same form. The maximum number of tasks in the Data Bank is limited by the dimension of the TOC array. The program currently provides for a maximum of 200 entries.

The overall flow diagram for the Data Bank Generator* (Routine TDBANK) is shown in Figure 6.2.** After setting up the Available Space List, the program reads a NAMELIST input that includes the value of flag NEW. When the value of NEW is left at zero, the program assumes that an existing bank is to be modified. Under this assumption, it reads the value of NTOC and the TOC array from the bank and then transfers all of the task descriptions from the permanent bank to drum storage (Section 5.3.2.3) before reading the incoming task description cards. The value of NEW is set equal to one when a

---

*A description of the job deck required to use the Data Bank Generator is presented in Section 9.1.

**The letters inscribed by circles in all of the flow diagrams represent points in the functional flow.

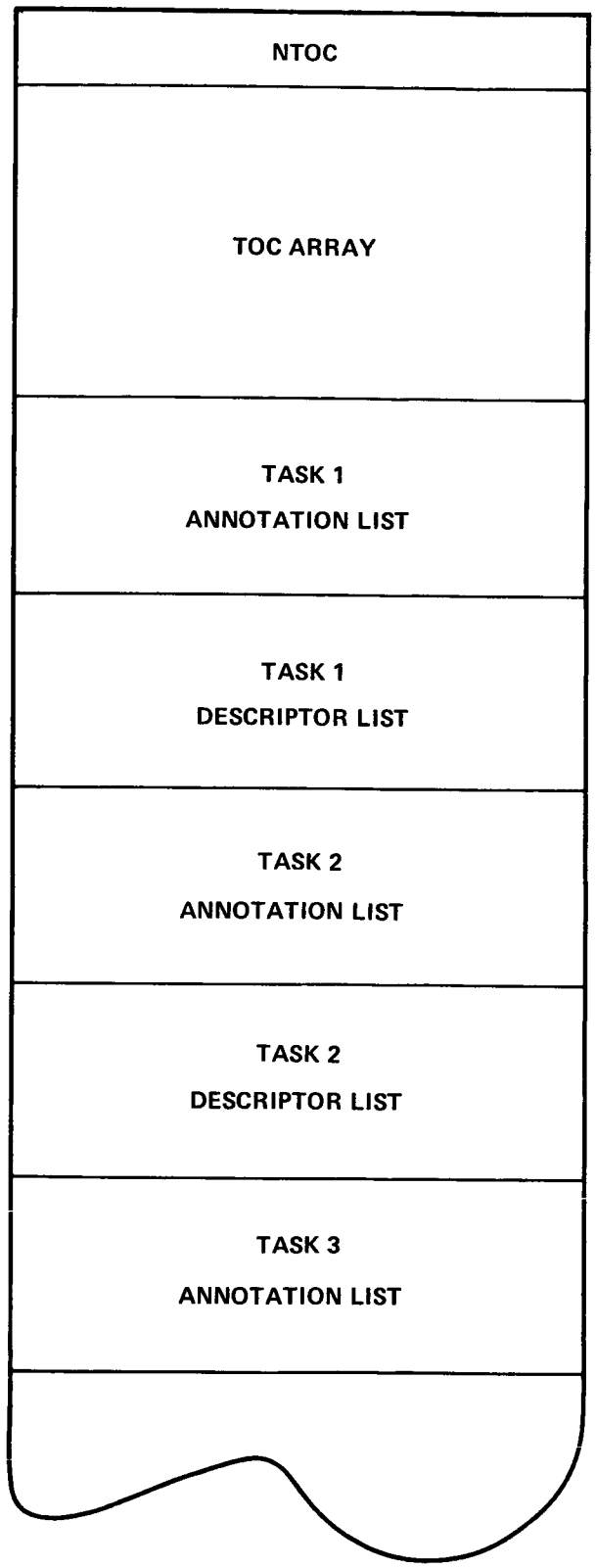


FIGURE 6.1 – FORMAT OF THE DATA BANK FILE.

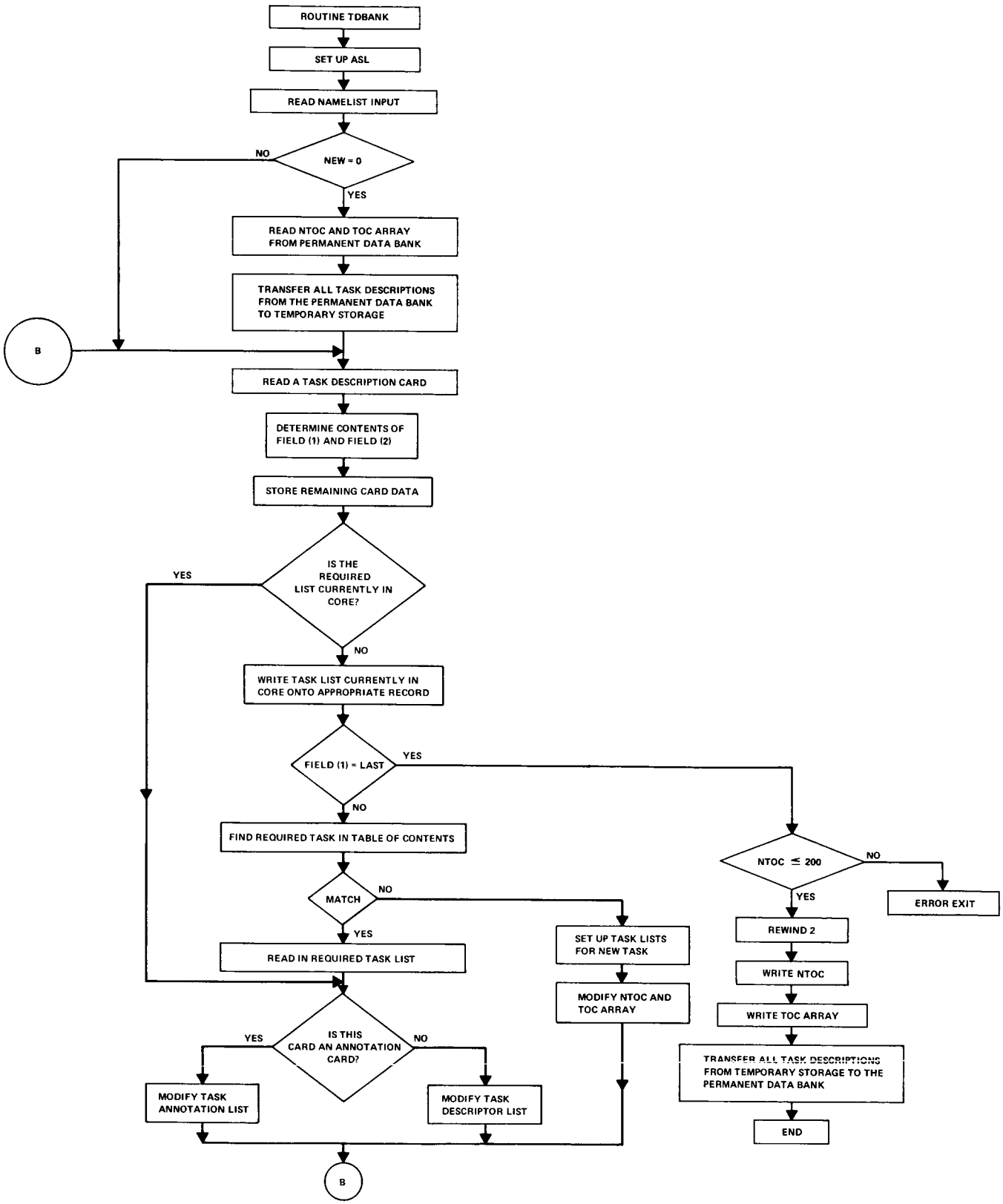


FIGURE 6.2 – OVERALL FLOW DIAGRAM FOR THE DATA BANK GENERATOR.

new bank is to be created. In this case the program proceeds directly to reading the task description cards.

All modifications specified by the incoming description cards are made to the task descriptions stored on the drum files. NTOC and the contents of the TOC array are also modified as necessary. When the modifications are complete (as indicated by the Last Card), a new data bank is created with the updated information.

The flow diagram for processing a Description Card is shown at Point A in Figure 6.2. When a card is read, the contents of the first two identifier fields are placed in the first two locations of Array FIELD. If the card is a Descriptor Card, the contents of the remaining fields are placed in successive locations in the FIELD array; if the card is an Annotation Card, the remaining information is placed, character by character, into Array COMENT. After the data on the card has been interpreted and stored, tests are made to determine if the list currently in core (i.e., the list required by the previous card) is the one needed by the current card. If so, the program transfers to the appropriate subsection where the modifications are made. If the list in core is not the required one, it is written out onto the appropriate record on the drum and a search is made of the task names in the TOC array to determine a match with the name stored in FIELD(1). When a match is found, the location of the corresponding record is obtained from the fourth column of the array for that entry, and the required list is retrieved from the drum file as described in Section 5.3.2. The program then transfers to the appropriate subsection to perform the modifications indicated on the card.*

The same methodology is used to modify the Annotation and Descriptor Lists. The information in the appropriate array (COMENT or FIELD) array is put into a list structure. The task sublists are searched until all of the identifier fields on the sublist match the respective fields in the new sublist. When a match is found, the old sublist is deleted or replaced with the new sublist as indicated. If no match is found, the sublist is added to the existing list structure.

If the name in FIELD(1) cannot be matched with a name in the TOC array, the program assumes that the name in

---

*See Reference 9.

FIELD(1) is the name of a new task. The identifier in FIELD(2) must then be 'TITLE', indicating a Title Card, or execution of the program will terminate with an error message.* If the card is a Title Card, the name is added to the TOC array, NTOC is increased by one, and the list containing the one Title Card is written out as the last record on the Annotation File. An empty record is also written out onto the Descriptor File to reserve that record for the task's descriptor list and hence preserve the parallelism between the two drum files.

There are two exceptions to this flow: the processing of Equivalence and Delete Cards. When the input card is an Equivalence Card, the TOC array is searched to determine a match with the name in the third identifier field on the card rather than FIELD(1). When the match is found, the Descriptor and Annotation Lists for that task are copied into core and modified by placing the name of the new task in the first cell of each list. The name of the new task is added to the TOC array, NTOC is increased by one, and both lists are written out as the last record on their respective drum files. Note that the lists for the new task are now separate and distinct from the corresponding lists for the original task. Subsequent modifications to either task description will have no effect upon the other.

When a task is to be deleted, NTOC is reduced by one and all subsequent entries in the TOC array are moved up one location, thus removing the reference to the deleted task. When the Last Card is encountered, the task descriptions will be transferred from the drum files to the Permanent Bank using the record numbers entered in the fourth column of the TOC array. Hence, the records for the deleted tasks will simply not be transferred. The final table of contents as well as the Annotation and Descriptor Lists for each task are printed out as the information is transferred to the Permanent Data Bank File, thus giving the user a complete description of the contents of the bank.

---

*Section A.1.1.

7.0 Functional Description of the Schedule Generator

The Schedule Generator is divided into four functional areas: executive control, initialization, window-finding, and scheduling. The executive control area exercises overall control of the program's execution while the initialization area controls the processing of all input data. Start-time windows for each task are determined by the window-finder. Finally, the scheduler section selects specific start-times from points within the start-time windows that are consistent with the task's performance objectives and "schedules" the task by updating the appropriate resource tables.

As stated above, the primary output of the Schedule Generator is a time history of the commitments for each resource and a list of start-times for each task. The data is printed out at the completion of the schedule and, on option, at regular intervals during the scheduling process. The frequency of the intermediate printout is specified by the user. Also on option, the program will print out a table illustrating the derivation of the task start-time windows thus enabling the user to observe the origins of the output data.

7.1 Executive Control

Figure 7.1 shows the overall flow diagram for the Schedule Generator* (Routine MAIN). The scheduling loop begins at Point A, immediately after initialization. The variable IPRIOR is incremented by one and a list is compiled (from a table of contents) of all tasks having a priority equal to IPRIOR. The tasks on this list are then considered for scheduling in the order of their occurrence.

After the descriptor list for the task in question has been retrieved from the auxiliary storage file, three preliminary tests are made. First, the quantity of each resource specified on an Amount Card is checked to insure that there is a sufficient amount available to support the minimum number of performances required (NPERFQ). If there is an insufficient amount of any resource, the task cannot be scheduled and so is not considered further. Similarly, a check is made to insure that each independent task specified on Enable and Inhibit Cards has already been

---

*A description of the job deck required to use the Schedule Generator is presented in Section 9.2.

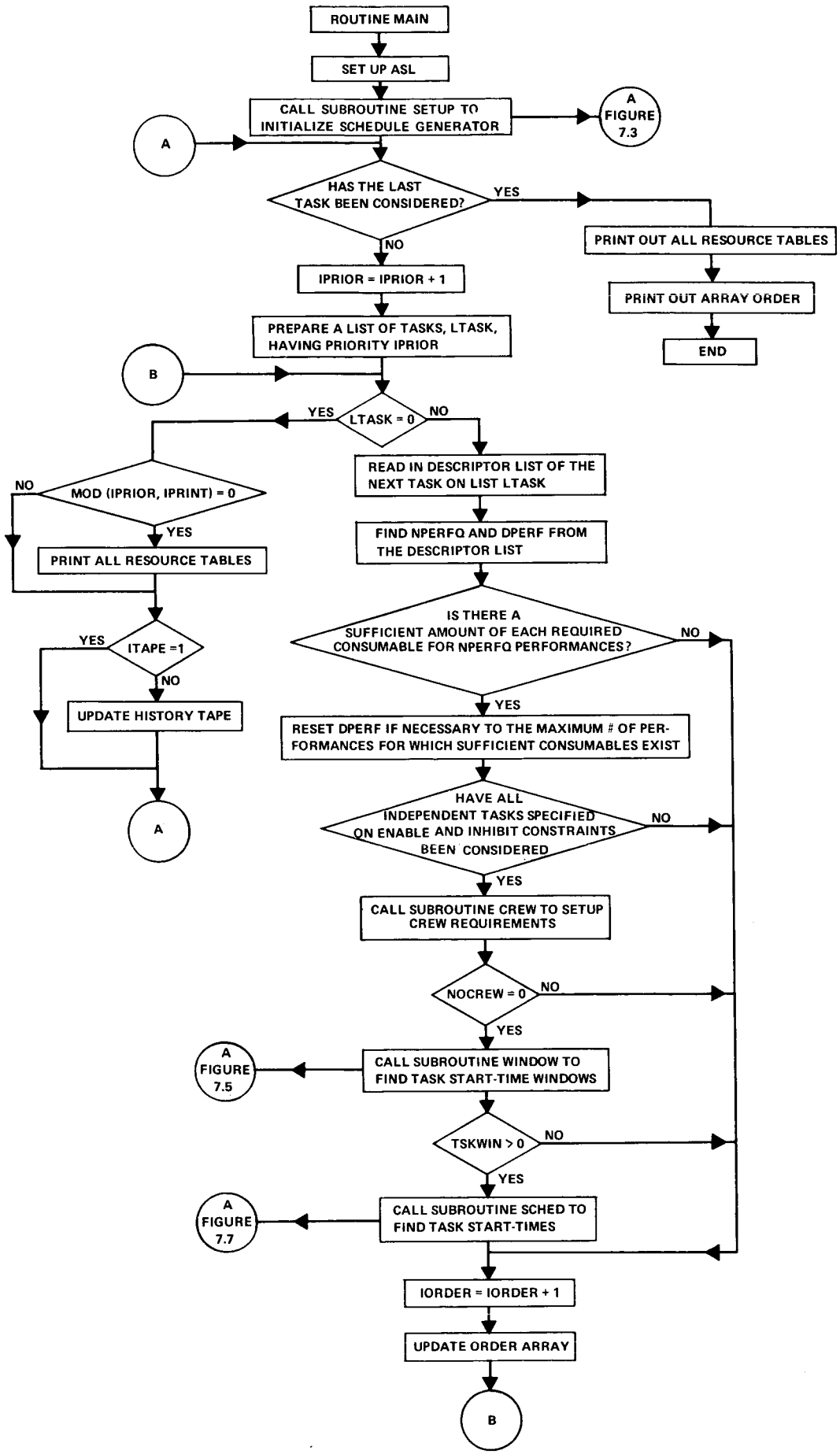


FIGURE 7.1 – SCHEDULE GENERATOR OVERALL FLOW DIAGRAM.

considered and that where applicable, the number of performances scheduled is not less than NPERFQ. Again, if these conditions are not satisfied, the task cannot be scheduled and is not considered further.

Crew assignments are made by Subroutine CREW. The subroutine replaces all skills named in the task Descriptor List with the name of the crewman who possesses that skill and then collects and counts the unassigned crewmen. The subroutine checks to insure that

1. The same crewman is not specified by name and skill, and

2. The number of unassigned crewmen is not less than the number of undesignated crewmen (i.e. ANY) required.

If either or both of these conditions cannot be met, the variable NOCREW is set equal to one indicating that the task cannot be scheduled.

When all of the preliminary tests are satisfied, the task start-time windows are determined by a call to Subroutine WINDOW, the executive program for the window-finder area. If one or more windows are found (TSKWIN>0), an attempt is made to schedule the task by calling Subroutine SCHED, the control program for the scheduler section. The output of Subroutine SCHED is a list of task start-times. The variable IORDER, representing the number of tasks processed, is then increased by one. The task name and the address of the start-time list are inserted as the IORDER entry in the ORDER array (columns 1 and 2 respectively). If no performances are scheduled, a zero is entered in place of the list address.

When all of the tasks at the priority level have been considered, all of the resource tables and the working arrays are written out onto the history tape, if the input variable ITAPE is equal to zero. The resource tables are also printed out whenever the variable IPRIOR is an exact multiple of the input Print Frequency Flag, IPRINT.

7.1.1 The History Tape

The format of the History Tape is shown in Figure 7.2. A general header, written at the conclusion of the initialization process (Section 7.2), includes all pertinent initialization data; i.e., the entire contents

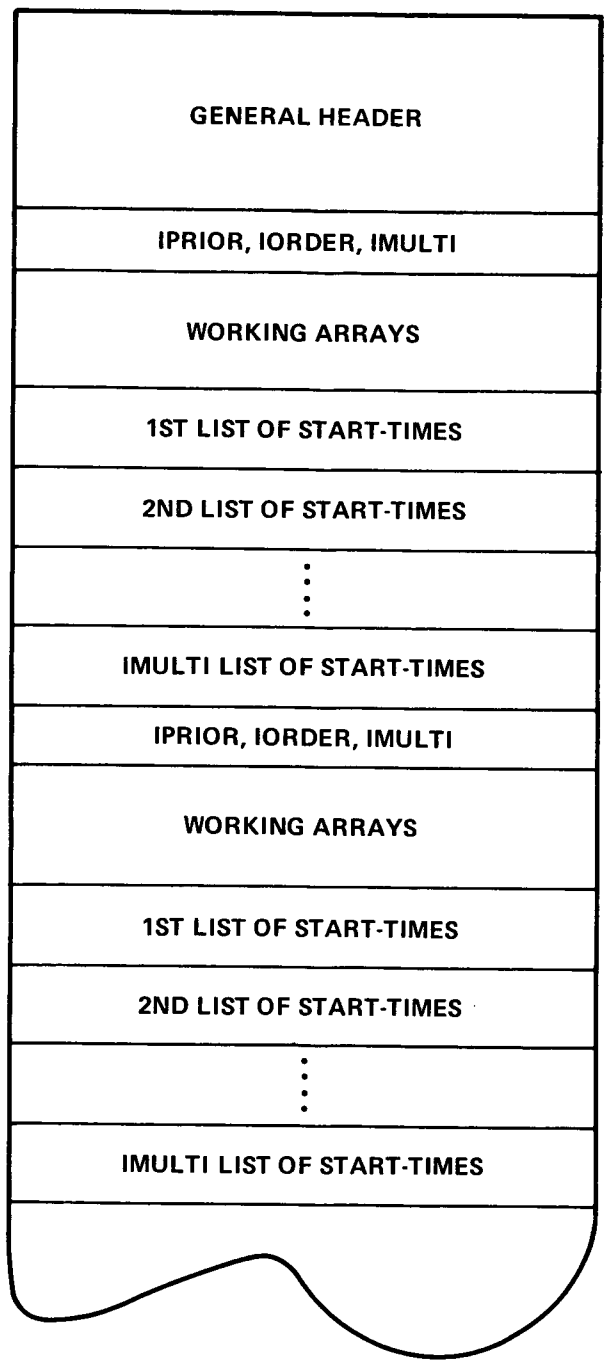


FIGURE 7.2 – HISTORY TAPE FORMAT.

of the WA array (including ephemeris data), the task table of contents, the initial configuration of the working arrays, and the final configurations of the task descriptions. The intermediate data, written at every priority level, contains the resource commitment tables, the working arrays (ORDER, LTABLE, etc.) and the symbolic string representation of every list of task start-times.

The intermediate data is written out in IMULTI+2 separate records, where the variable IMULTI is the total number of start-time lists whose addresses are entered in the ORDER array. The first record of the set contains the values of IPRIOR, IMULTI, and IORDER, the total number of entries in the ORDER array. The working arrays are written out onto the second record and the remaining IMULTI records contain the symbolic string representation of the start-time lists in the order of their appearance in the ORDER array.

7.2 Initialization

All initialization procedures are controlled by Subroutine SETUP. The flow diagram for the subroutine, which is called directly by the executive, is shown in Figure 7.3.

The value of the input variable IPRIOR determines the source of the initialization data. A value of IPRIOR equal to zero indicates that a completely new schedule is to be generated. For this option SETUP calls Subroutine TABIN which initializes all data tables from inputs specified in the NAMELIST. TABIN makes the necessary entries in Arrays LTABLE (Section 5.1.2), DTABLE (an array containing the maximum quantities and usage rates for each consumable), and CRWSKL (an array containing the name of each crewman and his assigned skill). In addition, it reads ephemeris data directly into the WA array and allocates space in that array for each of the resource commitment tables specified in LTABLE. After initialization of the tables, the descriptions of the tasks specified in the NAMELIST input are transferred from the Permanent Data Bank to the

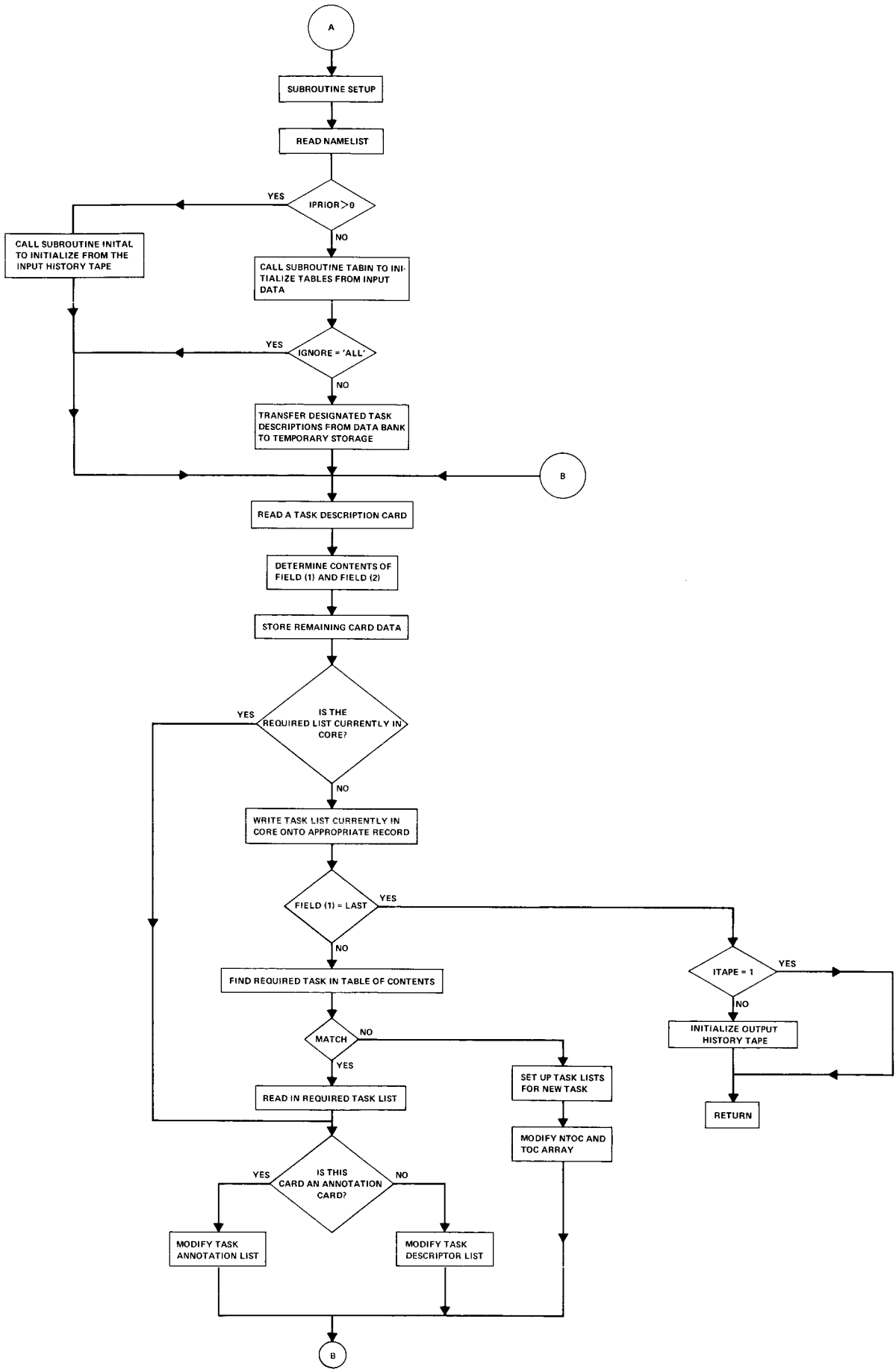


FIGURE 7.3 – FLOW DIAGRAM FOR THE SCHEDULE GENERATOR INITIALIZATION AREA.

temporary drum files. (If no descriptions are to be taken from the Data Bank, the variable IGNORE must be set equal to 'ALL' in the NAMELIST input.)

A value of IPRIOR greater than zero indicates that a previously generated schedule is to be completed and so the program must be initialized at some intermediate point with data from the History Tape. Under this option, SETUP calls Subroutine INITAL which reads in the general header and the working arrays as they appeared after all tasks at the IPRIOR priority level had been considered.

Additions and modifications to these task descriptions are accomplished in exactly the same way as in the Data Bank Generator (Figure 6.2) with one exception: the Delete Card cannot be used in the Schedule Generator. In the latter program, the task may be removed from scheduling consideration by setting the variable NPERFQ on the task's Objective Card equal to zero. The initialization process is finished when the Last Card is encountered. At that point the input data and all of the task descriptions are written onto a new History Tape (unless inhibited by setting the variable ITAPE equal to one in the NAMELIST input) and control is transferred back to the executive area.

7.3 The Window-Finder

The window-finding process closely follows the basic algorithm described in Section 2.3; i.e., all start-time windows for the ith requirement (or level) are determined between the limits BLIMIT and ELIMIT, the endpoints of a start-time window defined for the i-1 level.

As illustrated in Figure 2.3, the determination of the next level at which start-time windows are computed depends upon the result of the computations at the current level. For example, the search for windows at Level C was attempted because an acceptable window was found at Level B. Similarly, computations at Level D were attempted because acceptable windows were found at Level C. However, when no windows were found at Level D for the window $C_1 - C_1'$, the

the new limits $C_2 - C_2$ were substituted and the computations for the Level D were repeated. If the window $D_1 - D_1'$ had not been found, the program would have had to return all the way to the Level A, substitute $A_2 - A_2'$ for the window limits, and then proceed down to the Level B again.

The uncertainty of which level will be required next necessitates that the data pertaining to all levels be equally accessible. To achieve random accessing of the sublist structures in the Descriptor List, the address of each requirement and constraint sublist (with the exception of Amount Requirements) is placed in sequential locations in the REQ array. Hence for the Sleep Descriptor List shown in Figure 5.6, REQ(1) would contain the address stored in the element field of cell 4, REQ(2) the address stored in the element field of cell 5, etc. Hence, access to the ith sublist can be quickly obtained through reference to a subscripted variable.

The endpoints of the corresponding start-time windows for each requirement level are stored in two parallel lists, one containing the lower values for each window at that level, the other containing the higher values for the same windows. The addresses of these lists are similarly stored in an array. The first column of the ith entry in array LVWIN contains the address of the list of lower values while the second column of the same entry contains the address of the list of higher values. The configuration is illustrated in Figure 7.4 for the start-time windows shown in Figure 2.3. Note that the figure does not show all of the endpoints. Each pair of endpoints is discarded as it is used to define the limiting values to the next level. The figure illustrates the contents of the array at the point when window $E_1 - E_1'$ is discovered. Hence, the endpoints for windows $A_1 - A_1'$, $B_1 - B_1'$, $C_1 - C_1'$, $C_2 - C_2'$, and $D_1 - D_1'$ have already been discarded.

Subroutine WINDOW exercises overall control of the window-finding process. A flow diagram for the subroutine is shown in Figure 7.5. The window-finding process begins after the addresses of descriptor sublists are entered in the REQ array and the variables BLIMIT, ELIMIT, and I (the level indicator) are initialized. Each sublist

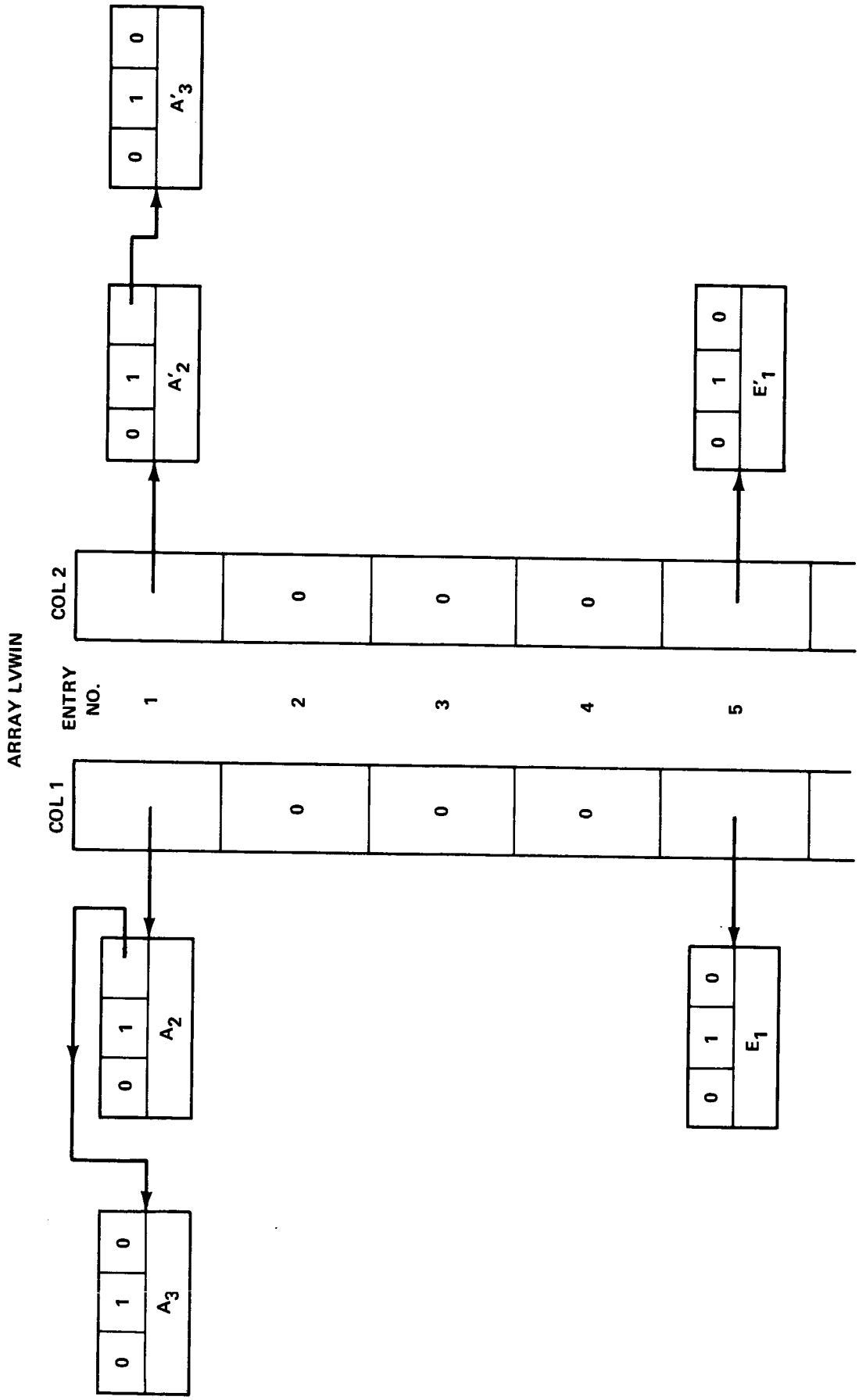


FIGURE 7.4 – STORAGE OF START-TIME WINDOWS.

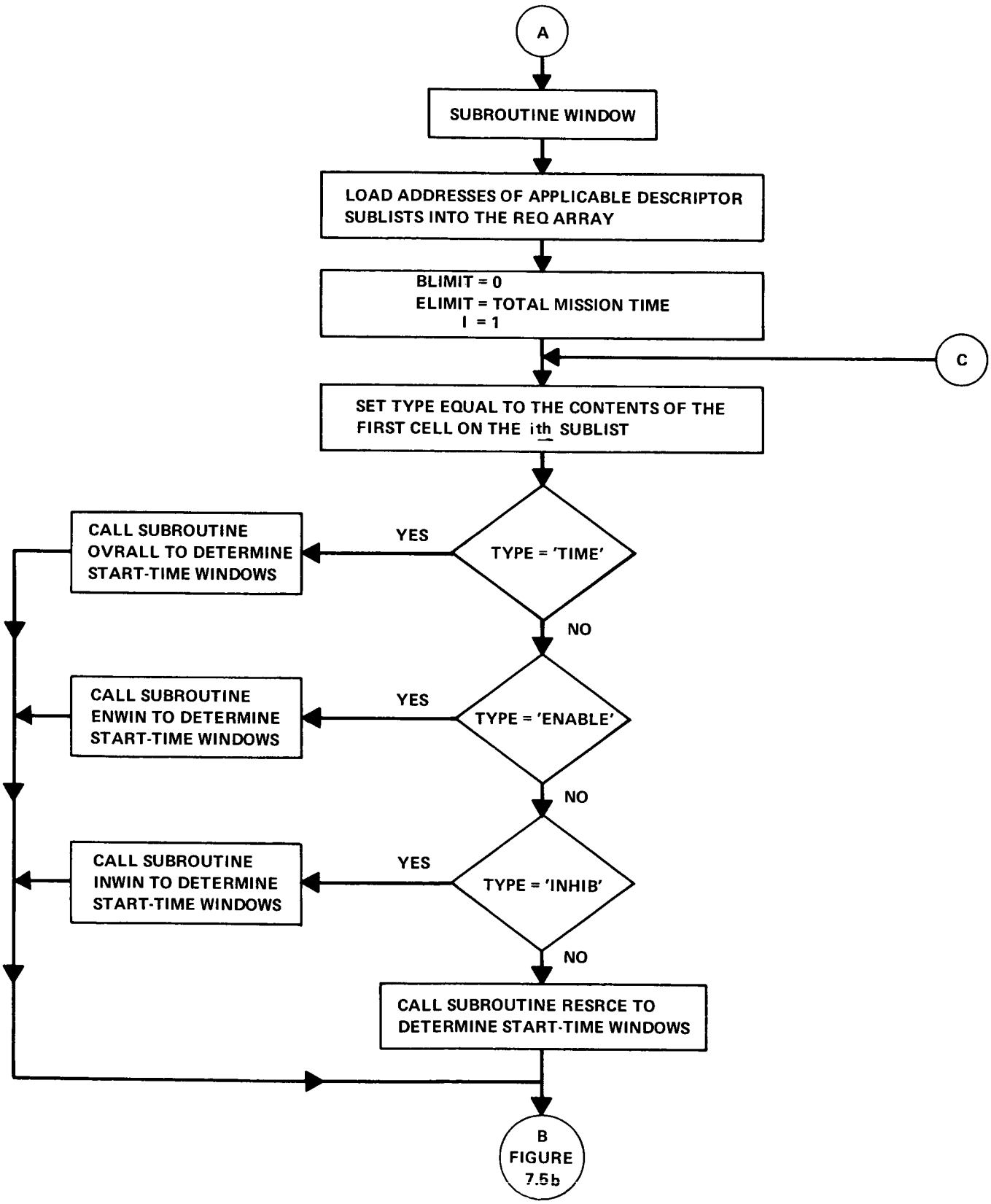


FIGURE 7.5a – FLOW DIAGRAM FOR THE WINDOW–FINDER AREA

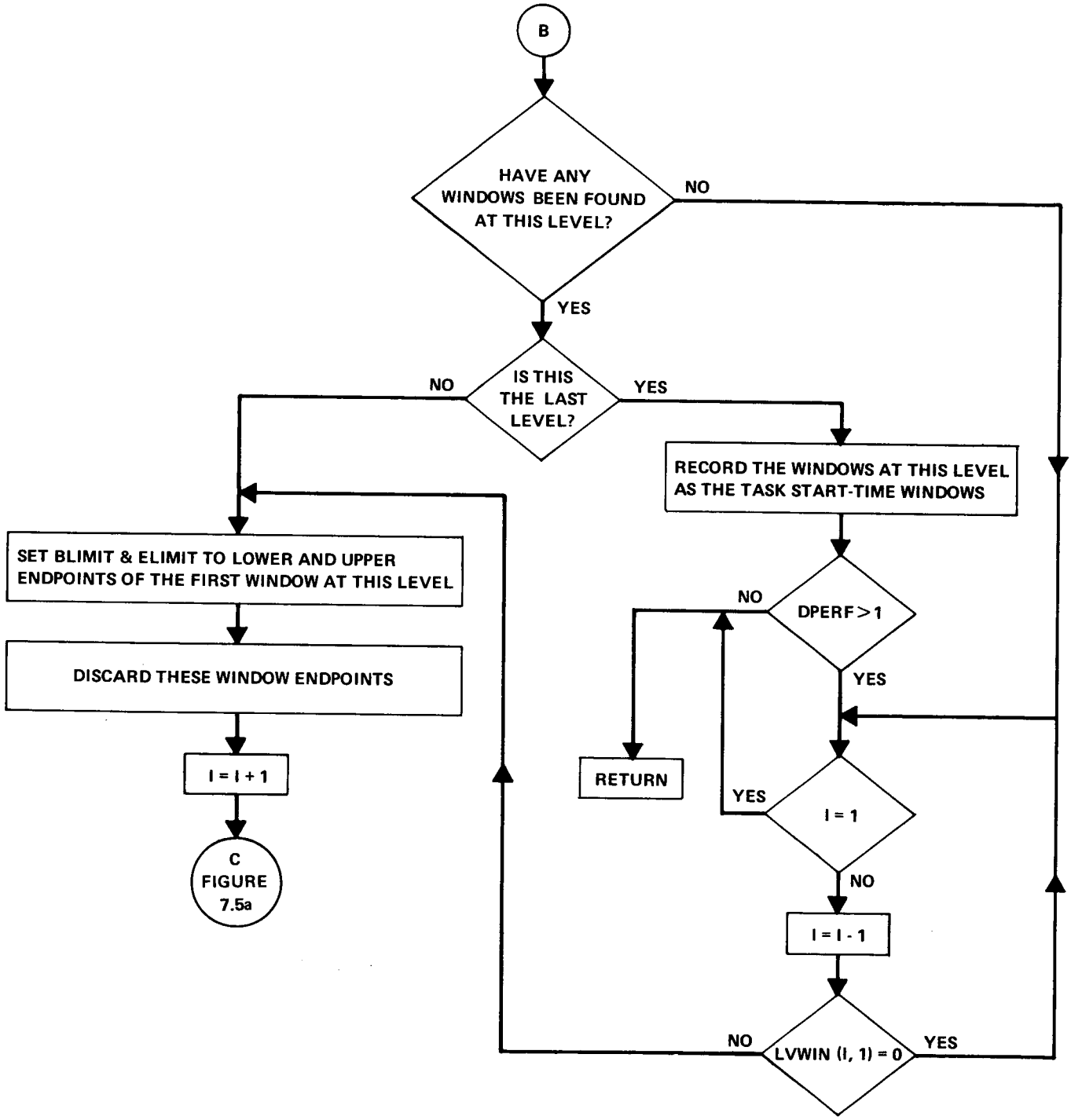


FIGURE 7.5b – FLOW DIAGRAM FOR THE WINDOW-FINDER AREA (CONTINUED).

is treated separately. The type designation is retrieved from the first cell of the sublist and the program transfers to one of four subroutines depending upon the contents of that cell: Subroutine OVRALL attempts to determine start-time windows for time constraints, Subroutine ENWIN is used for enable constraints, Subroutine INWIN for inhibit constraints, and Subroutine RESRCE for resource requirements. When control is returned to WINDOW, the requirement level indicator I is reset depending upon the results of the search for start-time windows. If acceptable windows have been found, the variables BLIMIT and ELIMIT are reset to the endpoints of the first window found at that level. The endpoints are removed from their respective lists, the level indicator is increased by one, and the process is repeated. When no windows are found, the level indicator is decreased until windows are found and the process is reinitiated from that point.

When the level indicator is at the last level, the windows at that level are recorded as acceptable start-time windows for the task. The windows are recorded in the complex list structure illustrated in Figure 7.6. Each sublist of the List TSKWIN contains all of the information relating to one start-time window. The first two cells contain the lower and upper endpoints of the window. If required, the remaining cells on the sublist contain the names of the crewmen selected by the program to fulfill the 'ANY' requirements. These names will not necessarily be the same for every window.

7.4 The Scheduler Area

The Scheduler Area selects start-times for individual task performances from points within the task's start-time windows. The start-times are selected to be consistent with the task's performance objectives. The output of the Scheduler Area is a list of start-times, START, the configuration of which is virtually identical to the configuration of the list of start-time windows, TSKWIN. In fact, there is only one difference: the first two cells on each sublist in TSKWIN (Figure 7.6) contain the endpoints of a task start-time window, while in list START, these two cells are replaced with one cell containing the actual start-time of the performance. As in TSKWIN, the remaining cells on the list, if any, contain the names of the crewmen selected to fulfill the 'ANY' requirements for that start-time.

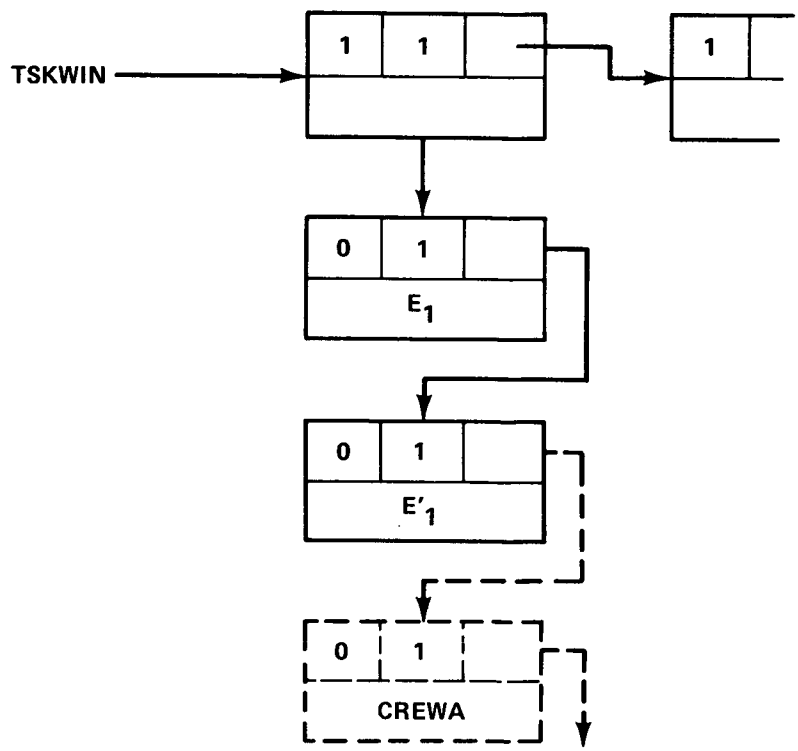


FIGURE 7.6 – CONFIGURATION OF THE LIST OF ACCEPTABLE TASK START-TIME WINDOWS AS OUTPUTTED FROM THE WINDOW-FINDER AREA.

Subroutine SCHED has overall control of the Scheduler Area. Its flow diagram is shown in Figure 7.7. If only one performance of the task is to be scheduled (DPERF=1), the program selects the lower endpoint of the first start-time window on List TSKWIN. If more than one performance is desired, the 'Objective' sublist is interrogated further to determine how these multiple performances are to be spaced. If both a nominal time between performances (TIMBET) and a tolerance on that time (TOL) are specified, the program transfers control to Subroutine MULTI which determines the number of performances (NPER) that can be scheduled consistent with the specified performance objectives. If NPER is less than NPERFQ, the minimum number required, the task cannot be scheduled. If NPERF$\geq$NPERFQ, control is transferred to Subroutine STIME which selects the actual start-times for NPER performances.

If TIMBET is not specified or is specified as a minimum, SCHED transfers control to Subroutine ASTART. ASTART selects the minimum time between performances as the greater of two quantities: the specified value of TIMBET or the time required for one performance of the task (defined as the length of time from the earliest beginning of a resource requirement to the latest end time of a resource requirement). The subroutine then selects as many start-times as possible to a maximum of DPERF. Again the task will not be scheduled if NPER<NPERFQ. If the task is to be scheduled, the program transfers to Subroutine ENTRY which updates all of the required resource tables and decreases the appropriate amount of each required consumable from the total stored in array DTABLE.

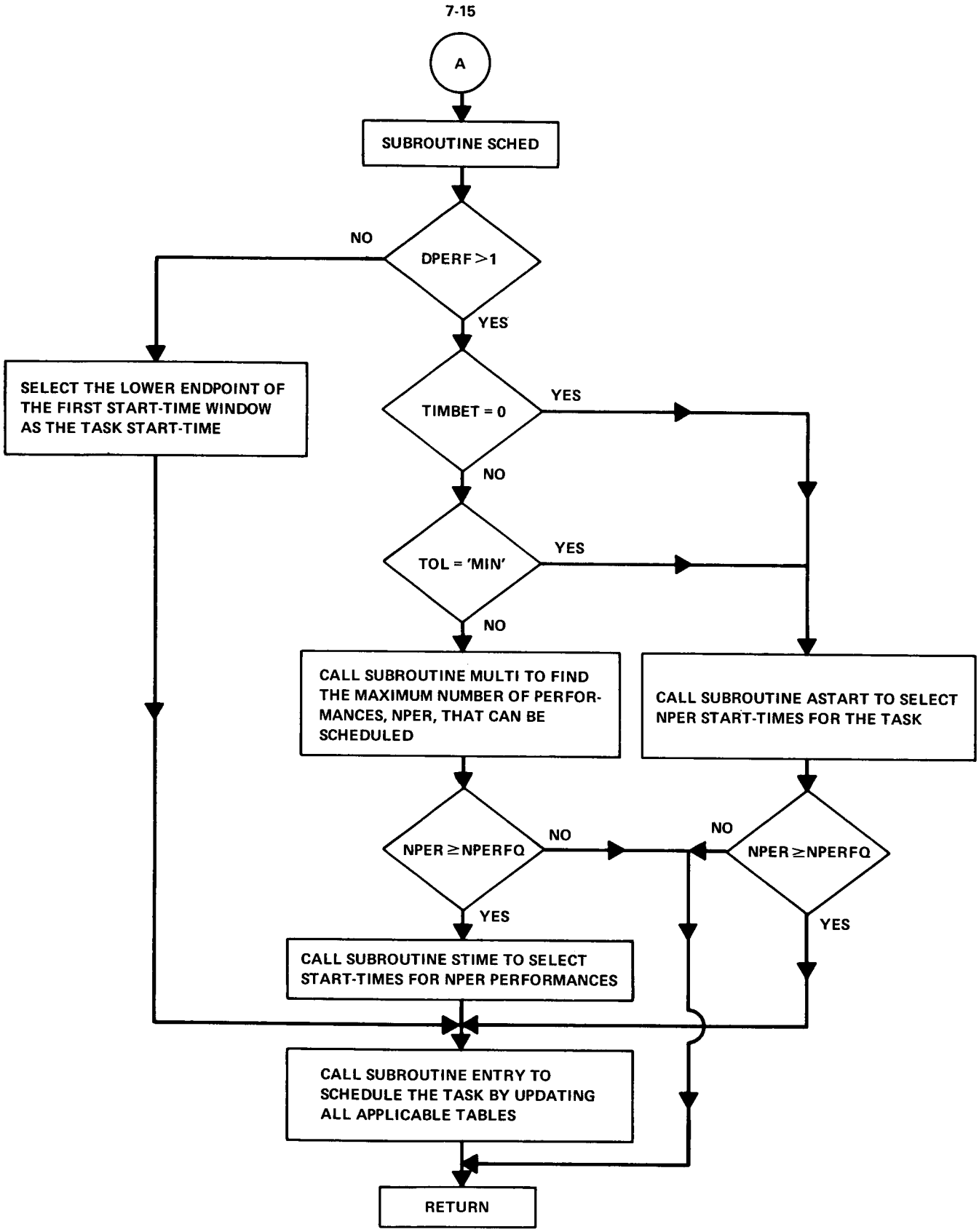


FIGURE 7.7 – FLOW DIAGRAM FOR THE SCHEDULER AREA

8.0 Functional Description of the ATS Data Processor

The Data Processor, like the Schedule Generator, is divided into four functional areas: executive control, initialization, coaxial (horizontal) plot generation, and periodic (vertical) plot generation.* As their names imply, the executive area exercises overall control of the program's execution, the initialization area controls the input of data from the History Tape, and the plot generation areas control the generation of the instructions and the selection of the appropriate data to produce the plot requested.

Plots generated by the Data Processor are actually created on the Stromberg-Carlson SC-4020 High-Speed Microfilm Recorder, a computer-driven plotter designed to operate as peripheral equipment to a high-speed digital computer. Each plot frame is first exhibited on the recorder's cathode ray tube. The tube face is then photographed and copies are reproduced on 35mm film or nine-inch sensitized vellum (7.5 inch square plot). Plots requiring more than one frame must be generated on a frame-by-frame basis.

The construction of the Data Processor is heavily influenced by the characteristics of the SC-4020 and its software library as well as the construction and operating characteristics of the AUPLOT system (References 11 and 12) which was used as an interface bewteen the ATS Data Processor and the SC-4020 software. Therefore, a discussion of the characteristics of these systems is presented in Section 8.1. Descriptions of the functional areas comprising the Data Processor follow in Sections 8.2 through 8.4.

8.1 Generation of Plots Using the SC-4020 and Auplot Systems

The SC-4020 is usually operated off-line from a magnetic tape input. The tape contains a series of commands to the plotter which are generated by a set of FORTRAN and assembly language computer programs designed specifically for this purpose (the SC-4020 and the accompanying software library are described in Reference 10). The programs enable the user to develop grid backgrounds, scale data, and print

---

*The terms "horizontal" and "vertical" refer to the direction in which the independent variable is plotted, referenced to the conventions established in Reference 10.

alphanumeric information in addition to plotting and connecting individual points on a curve. Though quite extensive, the software package requires the user to generate plot commands through a large number of complicated calls to specific-task subroutines (e.g. plot point, draw line, scale data point, etc.). Rather than use these routines directly, the construction of the Data Processor was significantly simplified by using the AUPLOT system as an interface between the programs in the Data Processor and those in the SC-4020 software library.

Though the AUPLOT system was designed as a general purpose interface between a user's program and any software library of a computer-driven plotter, its current version is implemented to interface specifically with the SC-4020 software library. The primary advantage to using AUPLOT is that while it also requires the user to generate a series of plot commands (subroutine calls), these commands are far more comprehensive and easier to implement than those permitted by the SC-4020 library. AUPLOT translates these comprehensive commands into one or more specific calls to the SC-4020 library routines.

8.1.1 Summary Description of the AUPLOT System

The following is a summary description of the basic features of the AUPLOT system and how they apply to the ATS Data Processor. A complete description of the system and its capabilities is presented in References 11 and 12.

The AUPLOT system is executed in two separate phases. The AUPLOT subprograms used in the first phase are called directly by the user's programs during execution. The AUPLOT subprograms place data tables and instructions (i.e., plot requests, legend information, scaling information, etc.) on an intermediate storage file, IOPLT. Phase 2, which is executed at the completion of the user's program during the same run, processes the instructions on file IOPLT in the same order in which they were placed on the file. Phase 2 generates output file PLOT using the subprograms from the SC-4020 library. This file, containing the specific instruction codes to drive the SC-4020, is stored on magnetic tape to be input to the plotter at some later time.

When AUPLOT is used, all communications between the user's program and the SC-4020 software library are made via the AUPLOT subprograms. A large number of instructions are available to the user from the standard AUPLOT instruction library. Each command is issued as a separate call to a specific subroutine in Phase 1 which places an instruction or a data point onto the IOPLT file. When processed during the execution of Phase 2, each instruction is routed to a specific subroutine (or group of subroutines) which calls programs in the SC-4020 library. Therefore, any extensions to the standard AUPLOT instruction library require the addition of two subroutines: the first to interface with the user's program during the execution of Phase 1 and a corresponding subroutine to interface with the SC-4020 library routines during the execution of Phase 2.

Three routines were added to the standard set of Phase 1 subprograms to permit the issuance of special labeling and scaling instructions. Four corresponding subroutines were added to the set of Phase 2 programs. (Instructions issued by one of the Phase 1 routines, Subroutine ABLIM, are processed by one of two subprograms in Phase 2 depending upon the setting of an indicator flag). Two additional modifications were made to the Phase 2 routines in order to provide the capability for shading on the periodic plots (Figures 3.3a and b). The standard AUPLOT Subroutine PLTXYQ, which is used to generate the commands for a first quadrant rectangular graph, was modified to enable it to determine when shading is to be performed and Subroutine SHADE was added to Phase 2 to generate specific shading instructions.

8.1.2 Collection of Graphical Data

The construction of a graph is a two-step process: the first step is the collection and storage of all data points; the second step is the mapping of the data points into an image space. In the first step, both coordinates of each data point must be collected. Furthermore, data points for the independent variable must be monotonically increasing. AUPLOT facilitates the collecting and storing of data by building data tables for each variable on a temporary storage file. The user is thus relieved of having to provide storage space for the data points in his own program.

Each data point is stored by a separate call to Subroutine COLECT. Thus, the statement

```
CALL COLECT (NAME, VALUE)
```

enters the number currently stored in the FORTRAN variable VALUE as the last entry in the table tagged with the

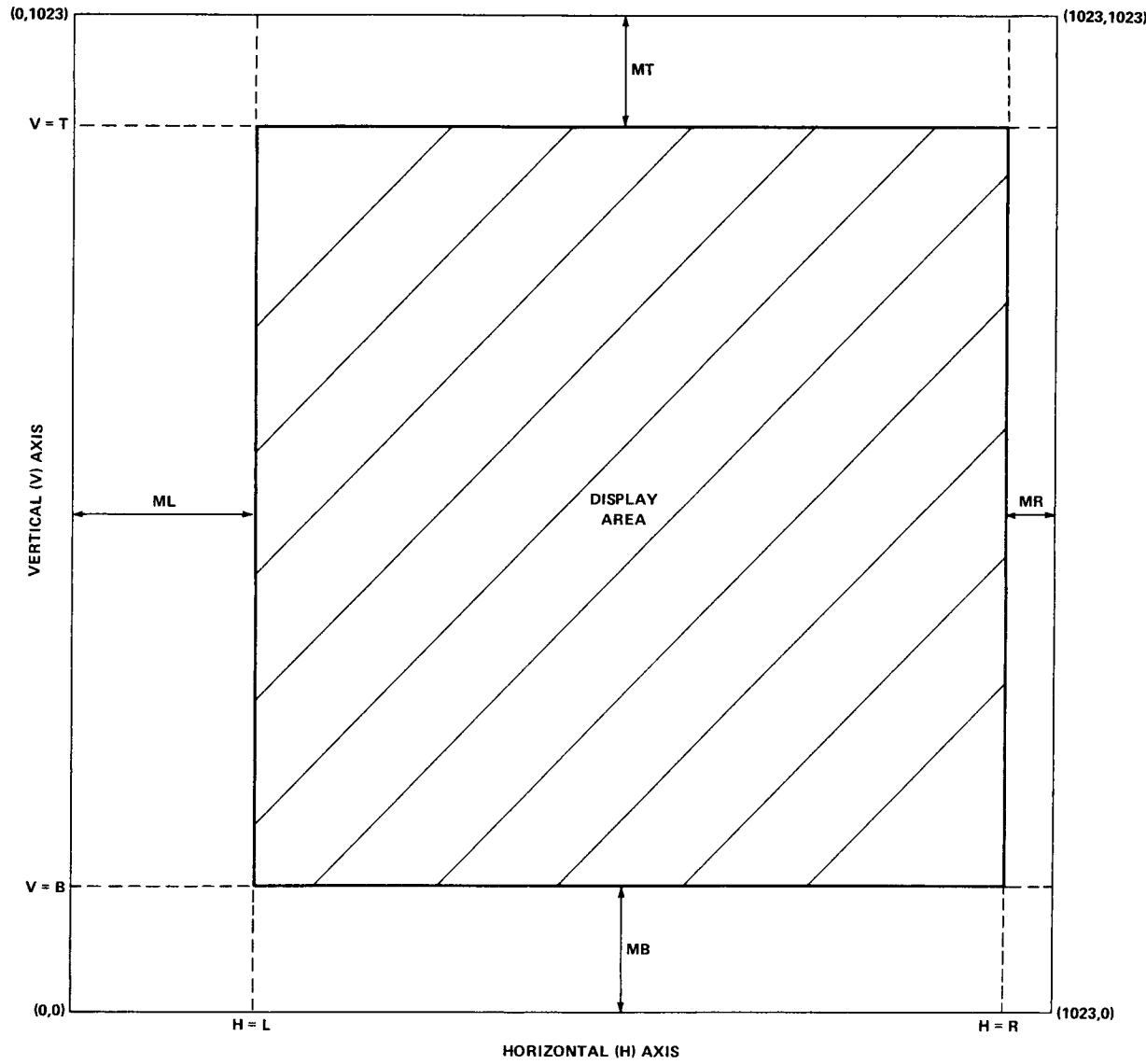
alphanumeric combination stored in the FORTRAN variable NAME. Note that all data must be stored as floating point numbers and that each variable must have a unique name. Thus, if two dependent variables X and Y are to be plotted against TIME, four data tables with tags TIME1, X, TIME2, and Y would be constructed. Mapping instructions would subsequently be issued to plot TIME1 vs X and TIME2 vs Y.

8.1.3 Plotting of Graphical Data

The image space of the SC-4020 is a first quadrant grid of 1024-horizontal by 1024-vertical raster points, with the origin in the lower left corner. The coordinates of each point must be specified to the plotter in terms of these raster points. Therefore, each data point must be appropriately scaled before it can be plotted. The scale factors are calculated by routines in the SC-4020 library. After setting aside space for the margins (which can be used for labeling information), the scale factors for each direction are calculated using the endpoints of the display area (in raster points) and the endpoints of the data range. Once the scale factors are established, the real values of each data point are scaled just before the point is plotted. If the value for either variable is outside the range defined by the endpoints for that variable, the point will not be plotted.

The relationships discussed above are illustrated in Figure 8.1. Suppose, for example, one wished to plot X vs Y in the display area in Figure 8.1. The horizontal scale factor would be computed so that the lowest value of X would be placed somewhere along the line X=L and the highest value along the line X=R. Similarly, the vertical scale factor would be calculated to place the lower and upper endpoints of the Y data range along the lines Y=B and Y=T respectively. All values between the endpoints would be plotted in the display area.

When two or more plots are to be superimposed on the same set of axes, the same factor will be used to scale all of the variables plotted in the same direction. The user must insure therefore that the data endpoints used to calculate the scale factors are of sufficient range to include all of the data for all of the variables. Similarly, graphs requiring more than one frame must be generated on a frame-by-frame basis. The scale factors must therefore be recalculated for each frame using the values of the data endpoints for that frame.



NOTE:

MB = BOTTOM IMAGE SPACE MARGIN
ML = LEFT IMAGE SPACE MARGIN
MR = RIGHT IMAGE SPACE MARGIN
MT = TOP IMAGE SPACE MARGIN

FIGURE 8.1 – SC-4020 GRID STRUCTURE.

8.2 Data Processor Executive Control and Initialization

The Data Processor is designed to graphically display the data produced by the Schedule Generator. The Data Processor has two sources of input data: punched cards and a History Tape produced by the Schedule Generator. The punched cards provide program control and plot description data. The latter includes the alphanumeric names of the variables to be plotted, the lower (TBEGIN) and upper (TEND) endpoints of the plot interval in mission elapsed time, and the scaling and labeling data needed to produce a finished graph. The History Tape provides the data base from which the plot data is obtained.

The Data Processor can generate as many plots as desired in a single run. However, each set of plot descriptions must be input separately. The program reads a set of data via a NAMELIST, performs the indicated calculations, and places the appropriate instructions onto the IOPLT file. The cycle is repeated until the variable LAST is set equal to one in the set of input data.

The overall flow diagram for the Data Processor* (Routine ATSPLT) is shown in Figure 8.2. The program is initialized from the History Tape at the priority level specified by the value of the variable IPRIOR in the NAMELIST.** The initialization is identical to that same function in the Schedule Generator (Section 7.2). Both programs use Subroutine INITAL to read and distribute the appropriate data from the History Tape to the proper core locations. The initialization procedure is omitted if the priority level specified in the input is the same as the one used for the previous plot. Under these circumstances, the appropriate data would already be stored in core.

The program transfers to the appropriate plot generation area on the basis of the contents of input arrays VSHADE, VPOINT, and HDEP. The arrays VSHADE and VPOINT contain the names of variables to be included on a vertical plot. VSHADE contains the names of the variables whose occurrences are to be plotted as shaded areas (maximum of

---

*A description of the job deck to use the Data Processor is presented in Section 9.3.

**Note that this method of initialization permits the display of the variables' status at any point in the scheduling process.

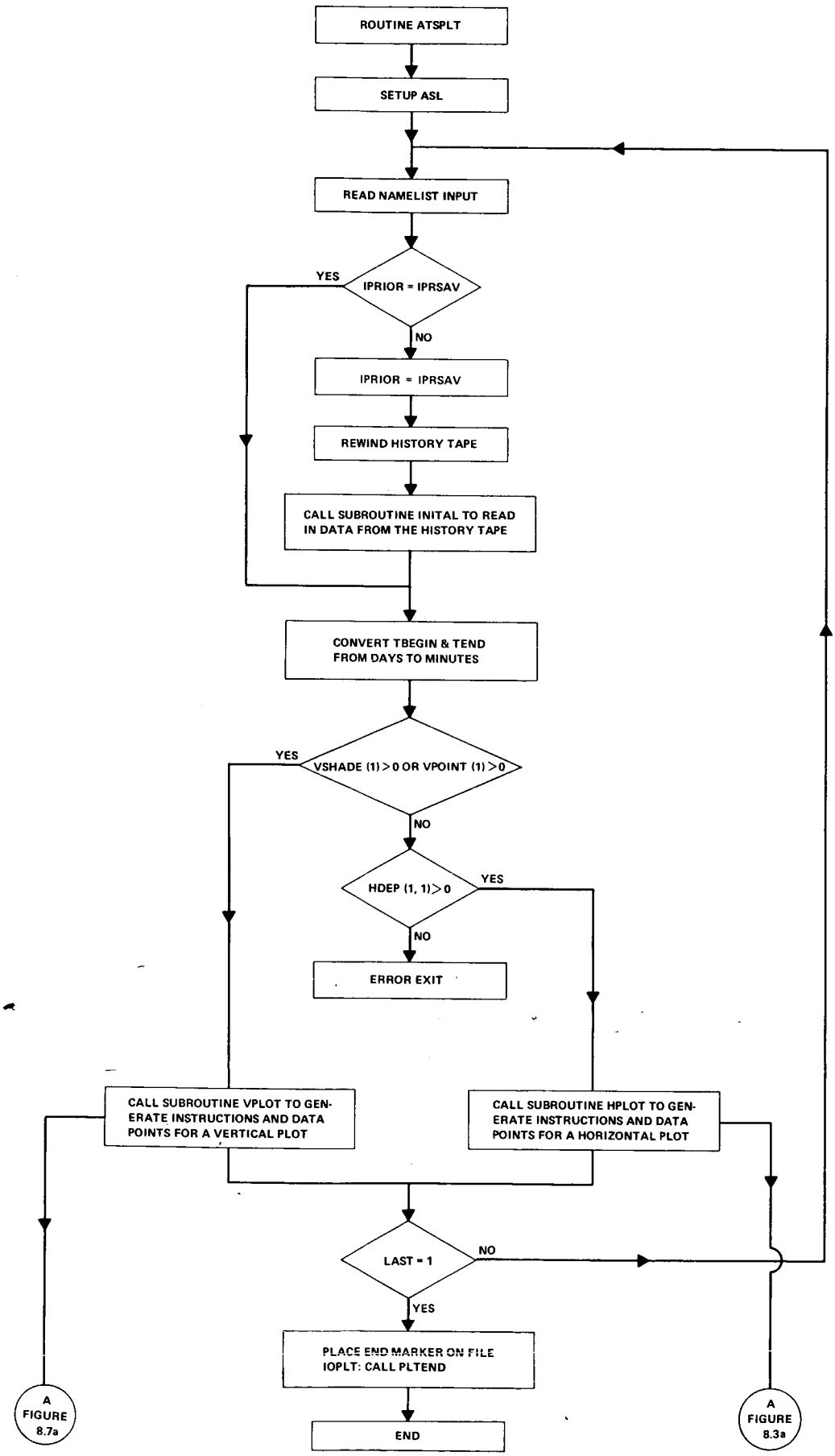


FIGURE 8.2 – DATA PROCESSOR OVERALL FLOW DIAGRAM.

three variables) while VPOINT contains the names of the variables whose occurrences are to be plotted as single points (maximum of 15 variables). If at least one of these arrays contains an entry, the program transfers to the vertical plot generation area.

Similarly, the first column of array HDEP contains the names of the variables to be included in a horizontal graph (maximum of five variables). If there is at least one entry in the HDEP array, the program transfers to the horizontal plot generation area. When control is returned to Routine ATSPLT, the cycle is repeated unless the variable LAST is equal to one. When the last plot has been processed, an end marker is placed on the IOPLT file and the execution is terminated.

8.3 Horizontal Plot Generation Area

The logical flow for the horizontal plot generation area (Subroutine HPLOT) is shown in Figures 8.3a and b. As the figures show, the flow follows the basic two-step process for the construction of a graph: collection and storage of data points followed by the generation of specific plot instructions to map the data into an image space.

8.3.1 Data Collection and Storage

Before any data points can be collected, ordinate limits for each variable must be established so that the graphs of each variable do not overlap. To establish these values, an arbitrary scale of 0.0 to 100.0 is used for the Y axis and the ordinate range for each variable is fixed at 10.0 (15.0 if the number of variables is less than three). After reserving the lower ten percent of the ordinate scale for the abscissa scale marks, the ordinate range between each variable is calculated from the relation

$$d = (90-nh)/(n+1)$$

where

d = ordinate range between each variable
n = number of variables to be plotted
h = ordinate range for each variable.

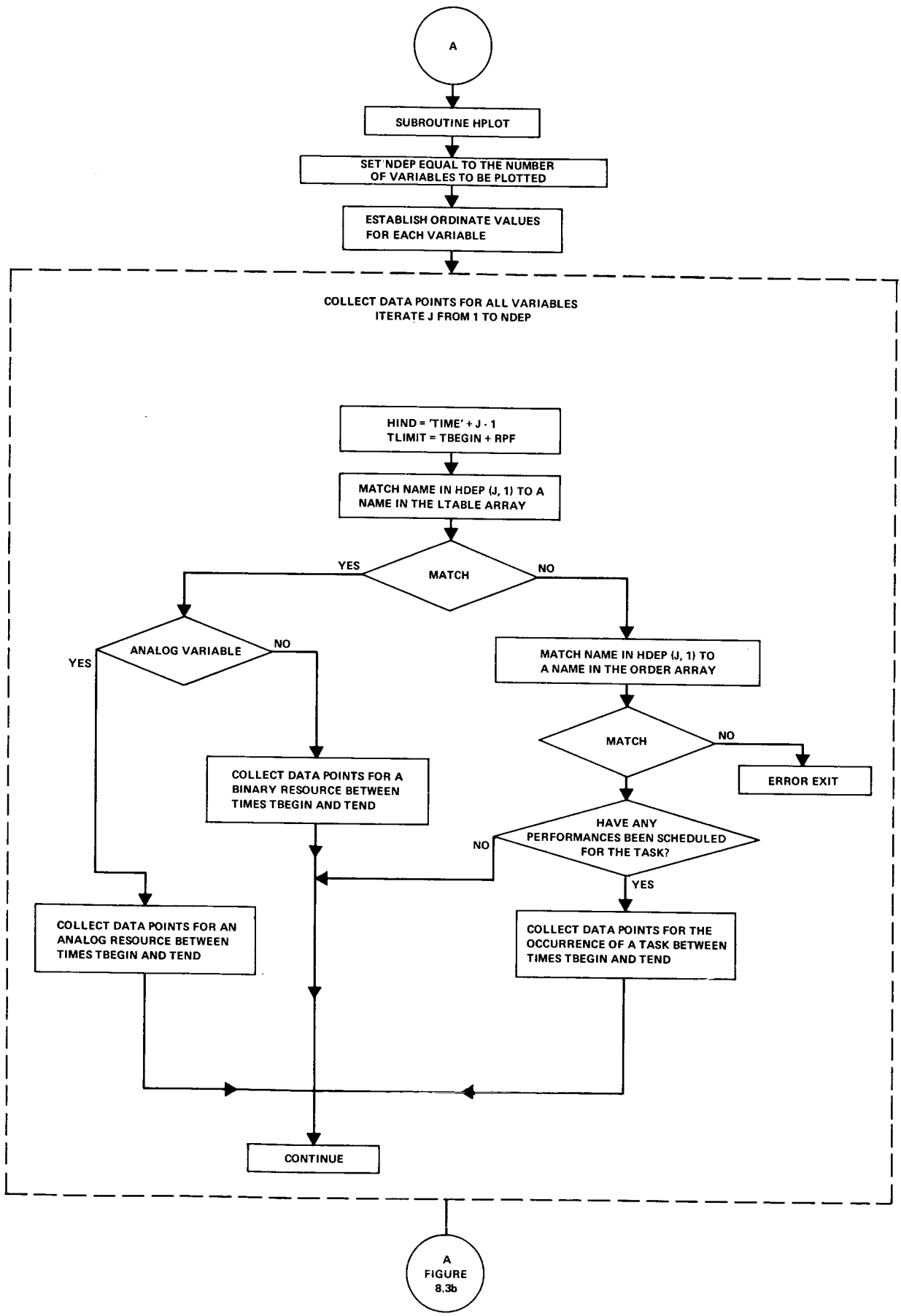


FIGURE 8.3a - FLOW DIAGRAM FOR THE HORIZONTAL PLOT GENERATION AREA
PART 1: DATA COLLECTION

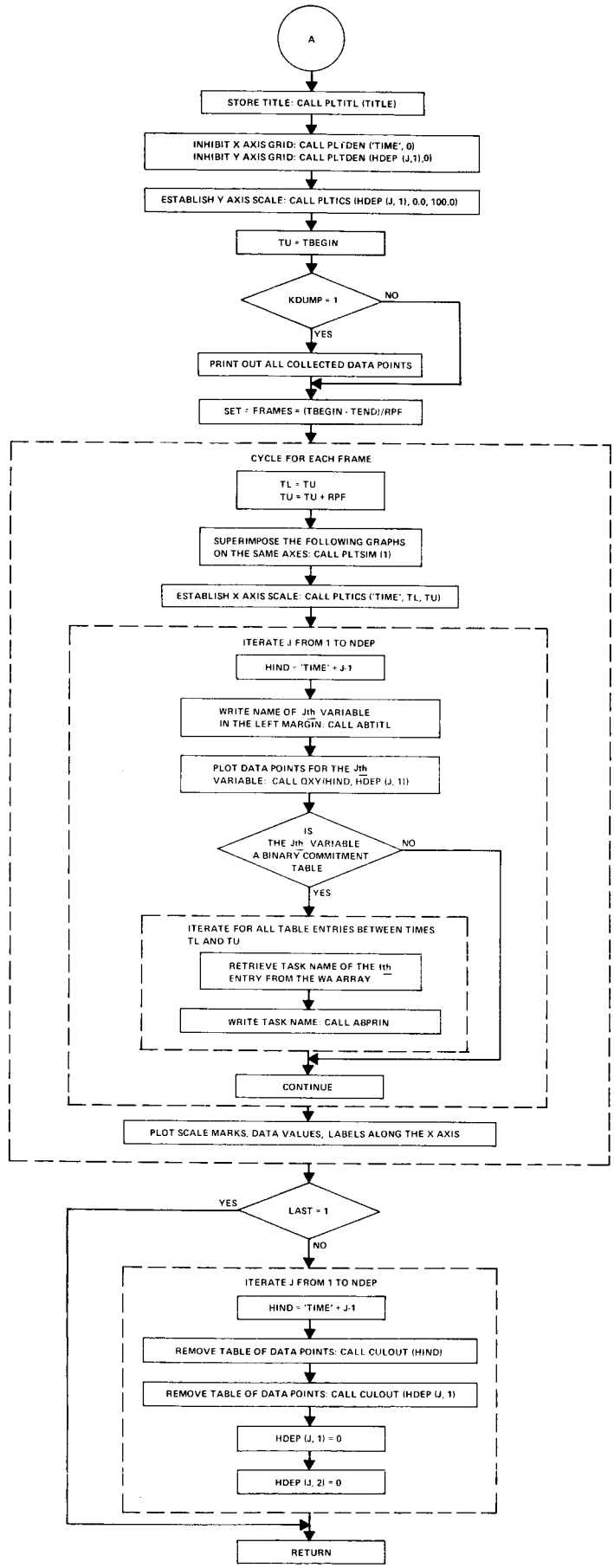


FIGURE 8.3b - FLOW DIAGRAM FOR THE HORIZONTAL PLOT GENERATION AREA
PART 2: GENERATIONS OF PLOT INSTURCTIONS

The lower and upper ordinate limits for each variable are calculated from the values of h and d and are entered in successive locations in Arrays LOWER and UPPER respectively. Thereafter, the kth variable in Array HDEP will have the ordinate limits LOWER(K) and UPPER(K).

The variables named in Array HDEP may be one of three types: a binary resource (including ephemeris), an analog resource, or a specific task. The names of the binary and analog resources must be specified exactly as they appear in Array LTABLE while the task name must be specified exactly as it appears in Array ORDER. As Figure 8.3a shows, the method of collecting data points depends upon the type of variable.

8.3.1.1 Collection of Data Points for a Binary Variable

The collection of data points for a binary variable is facilitated by the nature of the data itself, i.e., that it has only two states, 'off' and 'on'. The meaning of these states for each type of binary variable is shown in Table 8.1. On the horizontal plots, the 'off' state for each variable is plotted at the lower end of its ordinate range and the 'on' state at the high end of its range. Thus, in Figure 3.2b, the task LUNCHA occurs between 2.95 and 3.0 days and crewman CREWA is shown committed to that task over the same length of time.

Each entry in a binary commitment table (Tables 3.1 and 3.2a) is represented graphically by four data points as shown in Figure 8.4. The coordinates of all four points must be collected and stored in the sequence in which they occur. Thus, the sequence of call statements to store the ith entry in the table would be

Point 1
```
CALL COLECT(HIND, t_i1)
CALL COLECT(HDEP(K,1),LOWER(K))
```

Point 2
```
CALL COLECT(HIND, t_i1)
CALL COLECT(HDEP(K,1),UPPER(K))
```

Point 3
```
CALL COLECT(HIND, t_i2)
CALL COLECT(HDEP(K,1),UPPER(K))
```

Table 8.1

Interpretation of Binary Variable States

Variable Type	OFF	ON
Binary Resource Table	Uncommitted	Committed
Ephemeris Resource Table	Unavailable	Available
Task Performance	Performance not Occurring	Performance Occurring

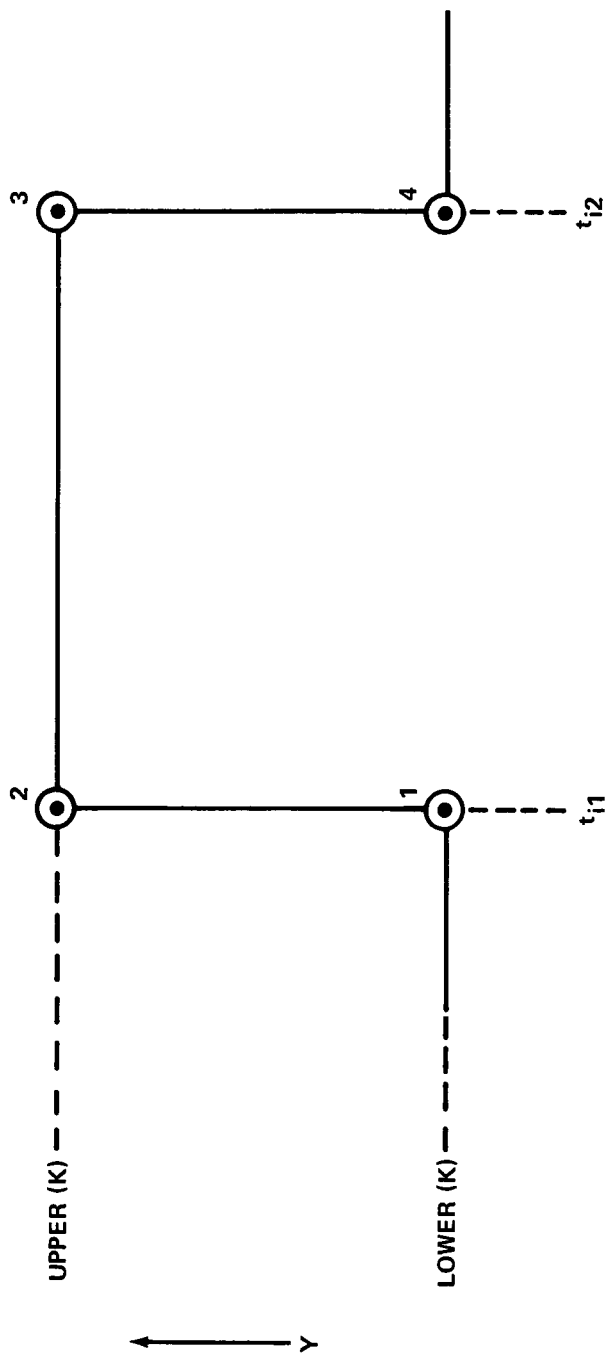


FIGURE 8.4 – GRAPHICAL REPRESENTATION OF THE i<u>th</u> ENTRY IN A BINARY COMMITMENT TABLE.

Point 4 { CALL COLECT (HIND, t_{i2})
CALL COLECT(HDEP(K,1),LOWER(K)).

As described above, the coordinates are stored in tables tagged with the variable name. During the execution of Phase 2, the coordinates of the point are retrieved by pairing successive entries in the appropriate tables. The coordinates are scaled, the point is plotted, and a line is drawn between successive points.

As noted above, the graphs are generated frame-by-frame, with the data range per frame (FORTRAN variable RPF) being specified by the user. In order to provide continuity between frames, the real data point corresponding to the rightmost end of the display area (line X=R in Figure 8.1) must be collected in the proper sequence along with the appropriate ordinate value. The abscissa value of the corresponding data point changes from frame to frame. However, the value is always known by first initializing the FORTRAN variable TLIMIT to RPF and then incrementing TLIMIT by RPF whenever the value of either endpoint in the table entry exceeds the current value of TLIMIT. When this occurs, the current value of TLIMIT is collected, along with the appropriate ordinate value, just prior to its being incremented. This cycle of collection and incrementation is repeated as many times as necessary until the next data point to be collected has a value of time less than the value of TLIMIT.

The two possibilities are illustrated in Figure 8.5. In Figure 8.5a, the ith and i+1 entries appear on either side of the current value of t_{limit}, indicating that they must appear in different frames. Therefore, the coordinates (t_{limit}, LOWER(K)) would be collected just prior to collecting the data points for the i+1 entry. In Figure 8.5b, the ith entry would bridge two adjacent frames and so the coordinates (t_{limit}, UPPER(K)) would be collected between the collection of points 2 and 3.

As shown in Figure 8.3a, a complete set of data points is collected for each variable in turn. TLIMIT is reinitialized to RPF immediately preceding the collection of each new set of points.

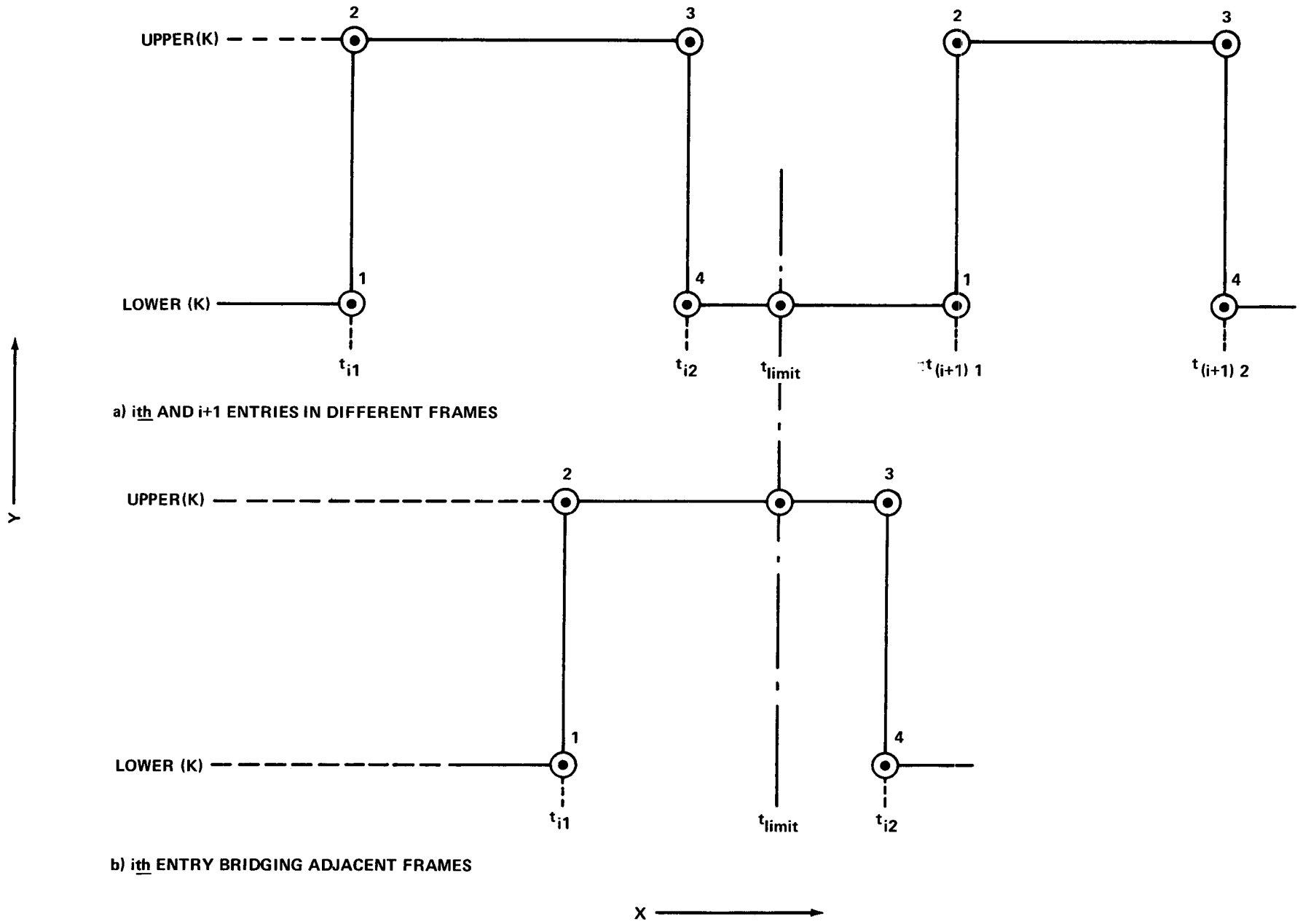


FIGURE 8.5 – COLLECTION OF DATA ENDPOINTS TO PROVIDE CONTINUITY BETWEEN FRAMES.

8.3.1.2 Collection of Data Points for an Analog Resource

The graphical representation of the analog resource table in Table 3.2b is shown in Figure 8.6. The ith entry in the table can be represented by two data points $(t_{i1}, R_{(i-1)2})$ and (t_{i1}, R_{i2}) in that order. The coordinates of these points are collected and stored by the call statements

CALL COLECT (HIND, t_{i1})

CALL COLECT (HDEP(K,1), $s(R_{(i-1)2})$)

CALL COLECT (HIND, t_{i1})

CALL COLECT (HDEP(K,1), $s(R_{i2})$)

where s is a scale factor that scales the ordinate to a value between LOWER(K) and UPPER(K). The scale factor is determined by first scanning all of the entries to be plotted to find R_L, the largest value of R_{i2}, and then calculating s from the relation

$$s = \frac{\text{UPPER(K)} - \text{LOWER(K)}}{R_L}$$

The remaining methodology is the same as for a binary resource. Thus, the points are processed in Phase 2 by pairing successive entries in the appropriate data tables to form a data point. The coordinates of the point are scaled, the point is plotted, and successive points are connected with a straight line.

Continuity between frames is also maintained in the same way as for the binary resources. Thus, when $t_{(i-1)1}$ and t_{i1} appear in different frames the data point $(t_{limit}, s(R_{(i-1)2})$ is collected and the value of TLIMIT is incremented by RPF. Again, the cycle of collection and incrementation is continued until $t_{i1} \leq t_{limit}$.

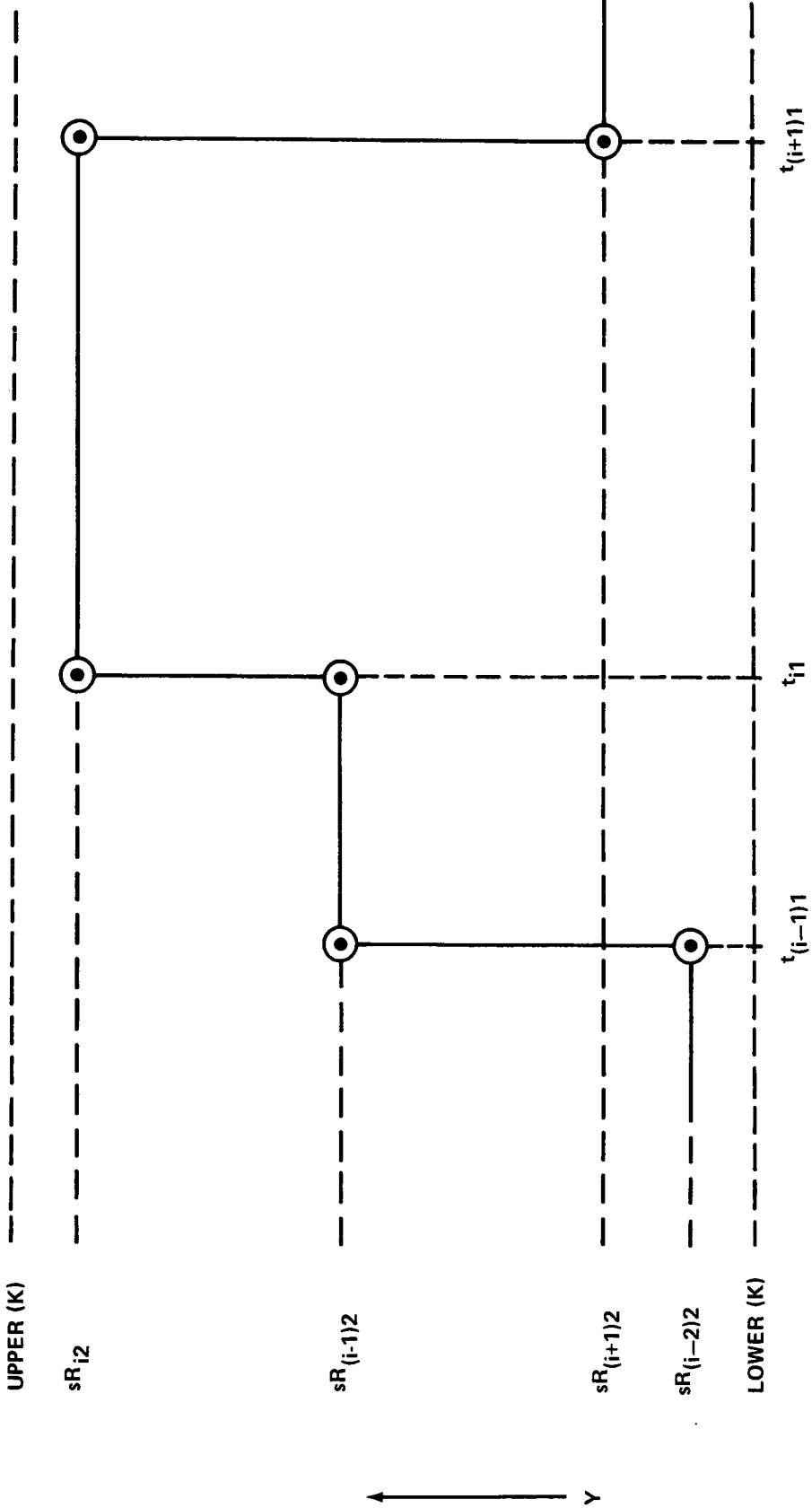


FIGURE 8.6 – GRAPHICAL REPRESENTATION OF AN ANALOG COMMITMENT TABLE.

8.3.1.3 Collection of Data Points to Represent the Performance of a Task

As noted in Table 8.1, task performances are also considered binary resources and, as such, are also represented graphically by four data points as shown in Figure 8.4. In this case, however, t_{i1} and t_{i2} are the beginning and end of the task performance, i.e., the earliest beginning of a resource (FORTRAN variable BEGIN) requirement and the latest end time of a resource requirement (FORTRAN variable END). When task occurrences are to be plotted, the Descriptor List for the task is read from the peripheral files and searched to determine the endpoints relative to the arbitrary task start-time. The variables BEGIN and END are determined for each task performance by adding the task start-time to the lower and upper endpoints determined from the task description. As described in Section 7.1, these start-times are in a list whose address is stored in the appropriate entry in the ORDER Array. Once determined, the data points for each performance are collected in the manner described in Section 8.3.1.1.

8.3.2 Plotting of the Horizontal Graphs

The flow diagram for the portion of Subroutine HPLOT concerned with plotting the data is shown in Figure 8.3b. As mentioned above, "plotting" consists of placing a series of instructions on the IOPLT file and, since the graphs must be generated frame-by-frame, the entire series of instructions must be repeated for every frame. The only exceptions are the instructions for grid backgrounds, labeling information, and ordinate scaling. They are issued only once for each plot since they do not change from frame to frame.

The first step in the instruction cycle is to establish the lower and upper data limits (FORTRAN variables TL and TU respectively) for the X axis. A call to AUPLOT Subroutine PLTSIM sets a flag which indicates that subsequent graphs are to be overlayed. Thereafter, all plot instructions issued between successive calls to Subroutine PLTSIM will produce graphs and labels on the same frame.

Plotting instructions are generated by successive calls to AUPLOT Subroutine QXY. Each call instructs that all data points between TL and TU for the variables named in the call statement be plotted on the display area. In addition, if the Y axis variable is a binary resource, the name of each task to which the resource is committed is retrieved from the

WA Array and is printed in the box representing that commitment. Before proceeding to the next frame, the scale marks, accompanying data values, and X axis labels are printed at the bottom of the frame. The scale marks are inserted at the endpoints of the display area and at seven equally spaced points in between.

When instructions for all of the frames have been completed, a series of instructions is issued to free each of the variable names for reuse. The statement

```
CALL CULOUT ('NAME')
```

causes the data tables for the variable 'NAME' to be erased from storage during the execution of Phase 2. These instructions will be processed in Phase 2 after the plots have been created and therefore will not affect their generation. New data tables with the same tag names can be placed on the IOPLT file subsequent to these instructions with no possibility of ambiguities with previous data.

8.4 Vertical Plot Generation Area

The vertical plots (Figure 3.3b) are used to highlight cyclic variations in the data as well as the interrelationships between different variables. Only binary variables, as defined by Table 8.1, may be plotted on a vertical plot; analog variables will be rejected.

Comparison of Figures 3.2b and 3.3b shows several significant differences between the horizontal and vertical plot formats. On the horizontal plots, the data for each variable is plotted over a different ordinate range so that the graphs never overlap. In contrast, the ordinate range on the vertical plots is the same for all variables over a given time period thus guaranteeing that the graphs of the different variables will overlap and so emphasize their interrelations. In order to differentiate between the different variables, each is plotted with a unique symbol or character. A variable may either be plotted as a box or a point. The box represents the duration of the 'on' state and is coded by shading for identification. A variable may also be plotted as a point. The point represents the

mid-point of the 'on' state for that variable.* As noted above, the user designates via the input data how each variable is to be plotted. The occurrence of those variables named in Array VSHADE will appear as shaded boxes while the occurrence of those designated in Array VPOINT will appear as points. The program selects unique symbols for each variable and prints out a legend (Figure 3.3a) defining these symbols before each plot.

Since the vertical plots use approximately the same ordinate range as one dependent variable on a horizontal plot, a large portion of the display area would remain blank when the graphs of the variables are overlayed. This blank space can be utilized however, by plotting successive time intervals one under the other. This format not only utilizes the entire display area (which reduces the number of frames required for the plots), but also increases the effectiveness of the display. Finally, in order to improve the continuity between frames, the axes for the independent variable and dependent variables are reversed, i.e., time is plotted along the ordinate (Y axis) and the variable states along the abscissa (X axis).

The resulting vertical graph in Figure 3.3b shows intervals of equal duration plotted along the Y axis. The value of time represented by the lower endpoint of each interval is shown adjacent to that endpoint. Note that the abscissa values representing the two states 'off' and 'on' are constant for any one interval but differ for different intervals, thus preventing the graphs of different variables from overlapping.

The flow diagrams for the vertical plot generation area (Subroutine VPLOT) are shown in Figures 8. a, b, and c. The logical flow follows the basic two-step plot construction process used for the horizontal plot area and so the flow diagrams for the two areas are quite similar.

8.4.1 Data Collection and Storage

The flow diagram for the collection and storage of data for vertical plots is shown in Figure 8.7a. Before any data points can be collected, the abscissa limits for each time interval on the frame must be established so that

---

*For task occurrences, the point represents the start-time of the task.

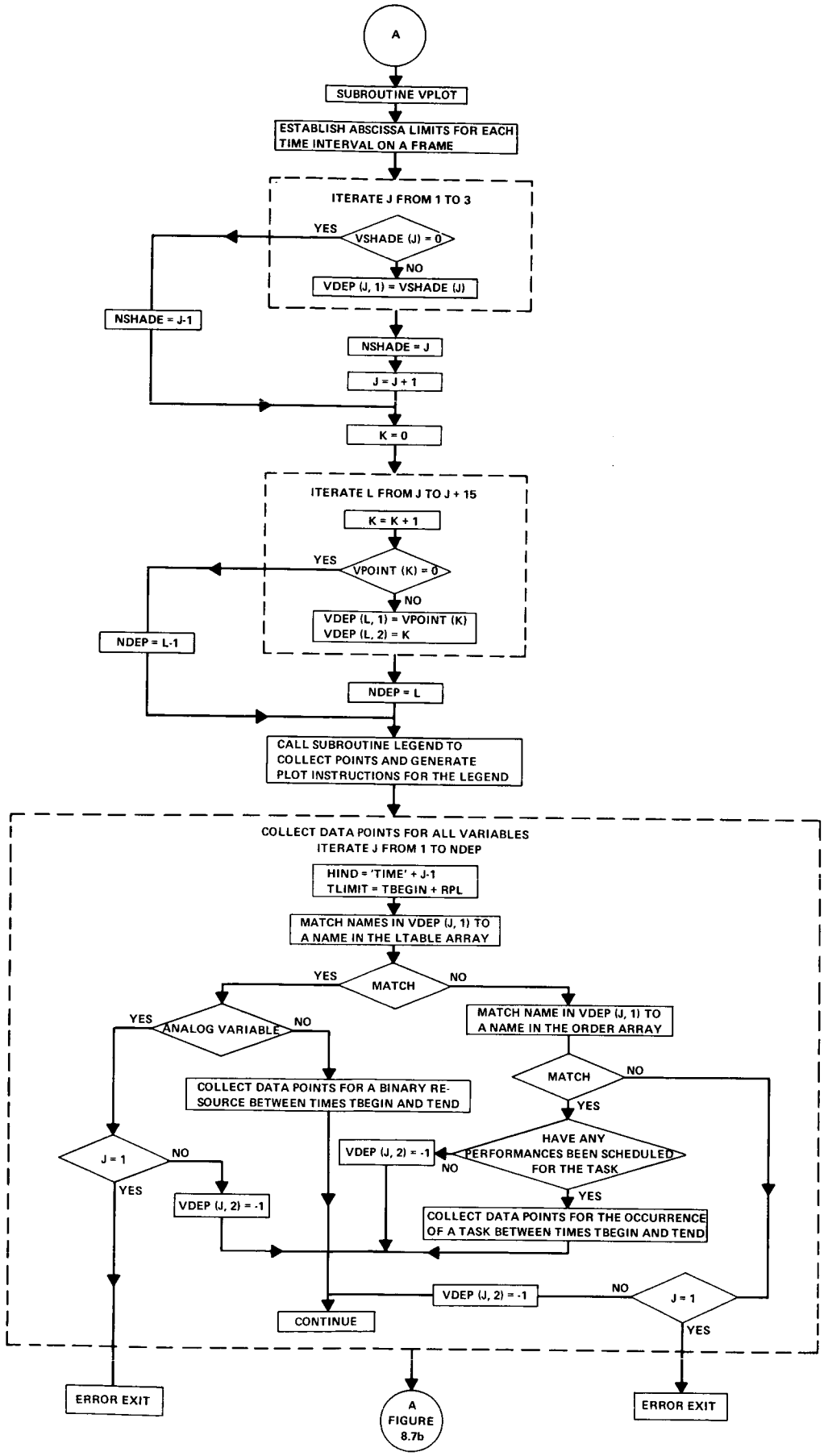


FIGURE 8.7a – FLOW DIAGRAM FOR THE VERTICAL PLOT GENERATION AREA
PART 1: DATA COLLECTION

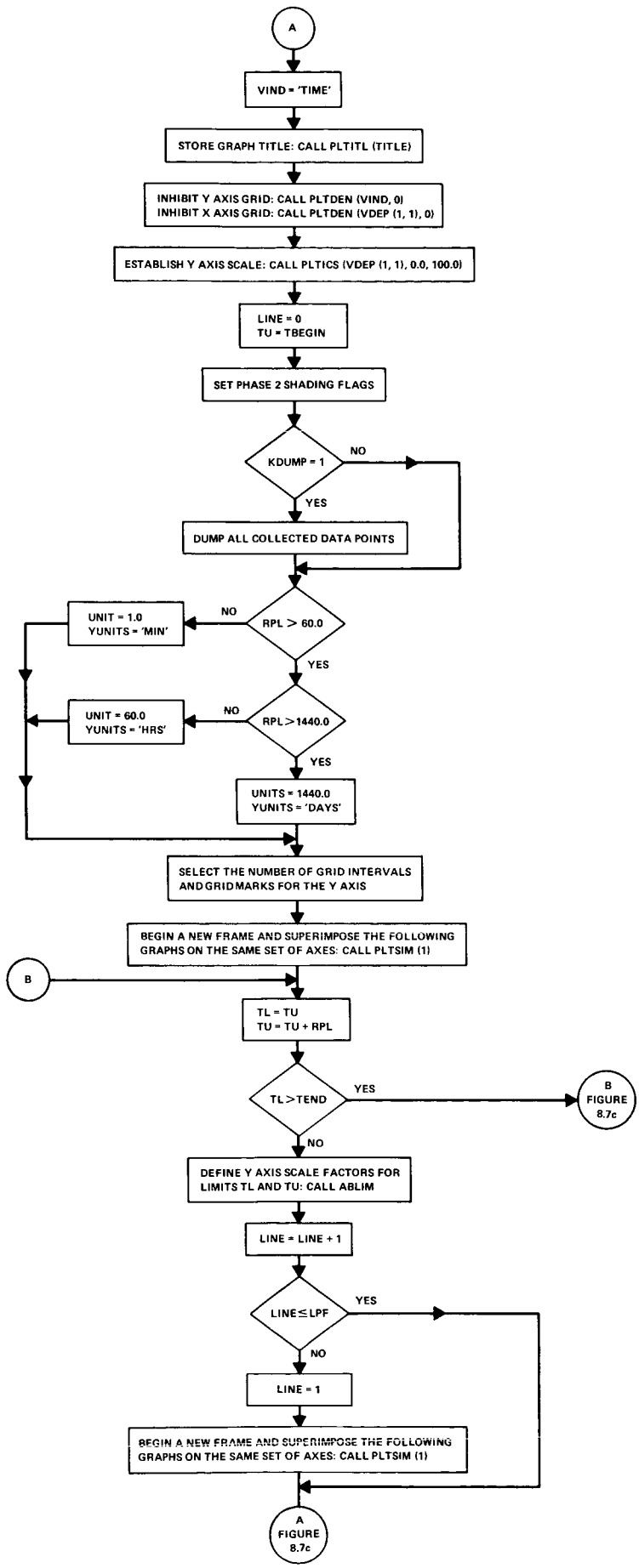


FIGURE 8.7b – FLOW DIAGRAM FOR THE VERTICAL PLOT GENERATION AREA
PART 2: GENERATION OF PLOT INSTRUCTIONS

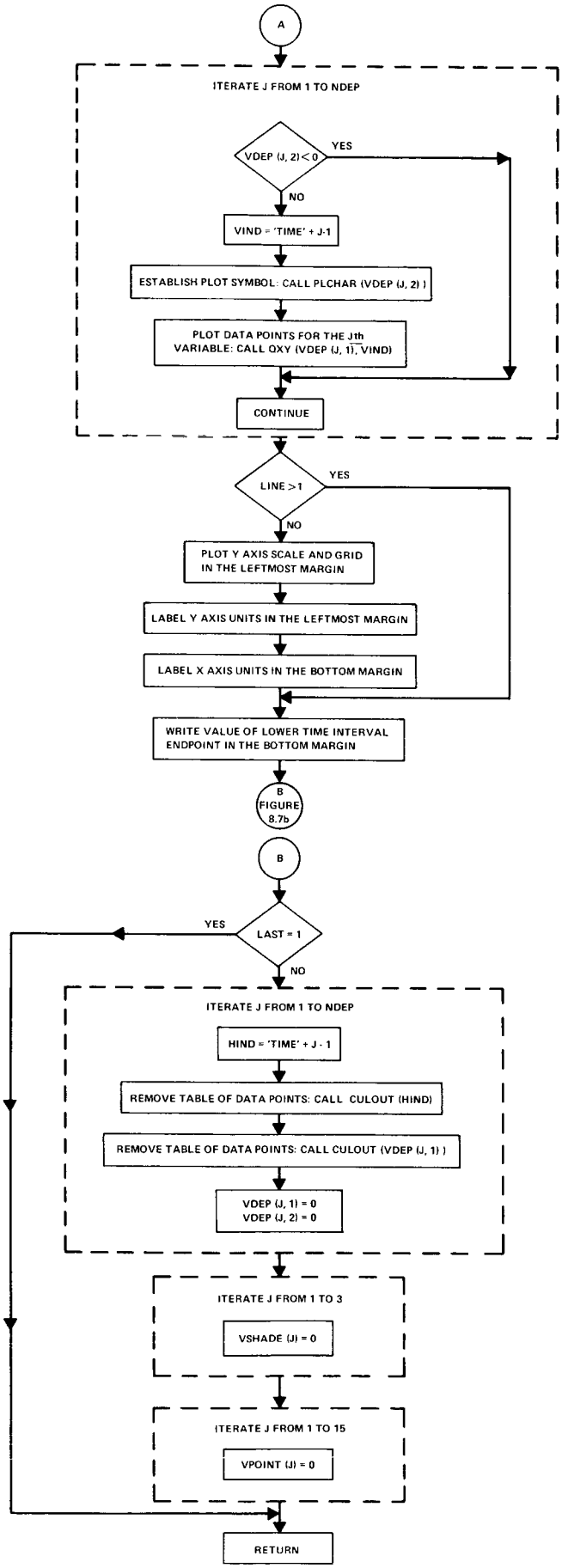


FIGURE 8.7c – FLOW DIAGRAM FOR THE VERTICAL PLOT GENERATION AREA
PART 2: GENERATION OF PLOT INSTUCTIONS (CONTINUED)

they do not overlap. The number of time intervals (or lines) to be plotted on one frame is specified in the NAMELIST input. Using an arbitrary abscissa scale of 0.0 to 100.0, the spacing between each interval is defined as

$$w = 100/L$$

and the abscissa range for each interval is defined as

$$h = w/3,$$

where

L = number of lines per frame (FORTRAN variable LPF)

w = abscissa spacing between successive time intervals (FORTRAN variable IWIDTH)

h = abscissa range for each time interval (FORTRAN variable HEIGHT).

The lower and upper abscissa scale limits for each time interval are calculated from the values of h and w and are entered in columns one and two respectively of Array HLIMIT. Thereafter, all data for the ith time interval (FORTRAN variable LINE) on every frame will be plotted between the limits HLIMIT (LINE,1) and HLIMIT (LINE,2).

As with the horizontal plots, any binary resources named in Arrays VSHADE and VPOINT must be specified exactly as they appear in Array LTABLE while each task named in the input arrays must be specified exactly as it appears in the ORDER Array. As shown in Figure 8.7a, the names in both arrays are transferred to sequential locations in the first column of Array VDEP as they are counted. The input sequence number is entered in the corresponding second column for those entries that appear in Array VPOINT. The method of data collection and plotting for each variable is determined by the contents of this second column. A zero entry specifies that the data is to be collected and plotted as a box while a non-zero entry stipulates that the data is to be collected and plotted as a point.

8.4.1.1 Collection of Data Points to Represent Binary Variables as Shaded Boxes

When a binary variable (i.e., binary resource table, binary ephemeris table, or the occurrence of a task) is to be plotted as a shaded box (Figure 3.3), the 'on' state for the variable is represented by four data points as shown in Figure 8.8. This representation is identical to the one shown in Figure 8.4 for the horizontal plots except for the orientation of the axes. In this case, t_{i1} and t_{i2} again represent the endpoints of the 'on' state; however, the 'off' and 'on' states are now represented by the two abscissa values HLIMIT(LINE,2) and HLIMIT(LINE,1) respectively. The sequence of call statements to collect these data points is the same as those shown above in Section 8.3.1.1.

Continuity is maintained between successive time intervals in the same manner as it is maintained between successive frames for the horizontal plots. The user specifies the range of the time interval (FORTRAN variable RPL). For each variable, TLIMIT is initialized to RPL and is then incremented by RPL whenever the value of either endpoint, t_{i1} or t_{i2}, exceeds the current value of TLIMIT. The value of LINE, the time interval counter, is also incremented but its value is reset to one whenever it exceeds the value of LPF, the number of intervals to be plotted on one frame. In addition, when it is the value of t_{i2} that exceeds the value of TLIMIT, the value of TLIMIT is collected along with the appropriate abscissa, HLIMIT(LINE,1), thus providing continuity of data between successive time intervals. Note that this method of maintaining continuity permits the user two degrees of control over the output graph: the length of the time interval (RPL) and the number of intervals to be placed on one frame (LPF).

The meaning of the 'on' state for the different types of variables is the same as for the horizontal plots (Table 8.1). The endpoints of the 'on' state (t_{i1} and t_{i2} in Figure 8.8) for a binary resource or binary ephemeris table are the pair of points comprising one entry in the table. For an occurrence of a task, the endpoints represent the beginning and end of a performance. The method of determining these endpoints for a task occurrence is described in Section 8.3.1.3.

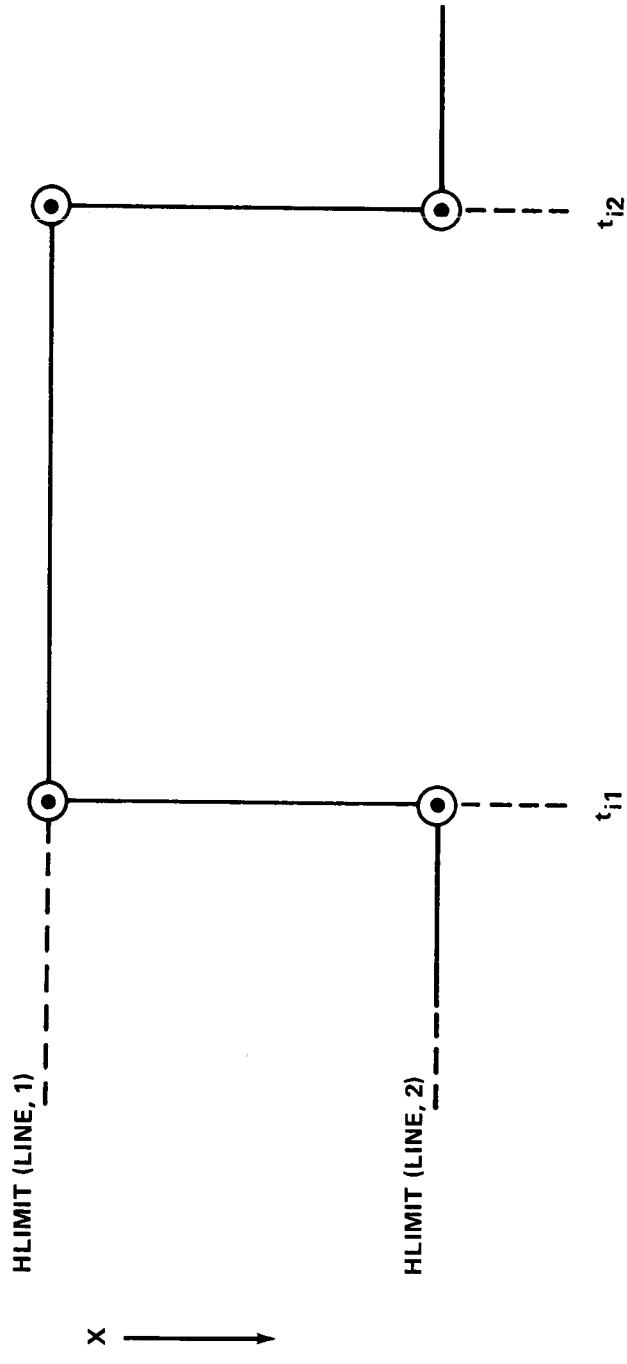


FIGURE 8.8 – GRAPHICAL REPRESENTATION OF THE 'ON' STATE FOR A VERTICAL PLOT.

8.4.1.2 Collection of Data Points to Represent Binary Variables as Points

As noted above, when the 'on' state of a binary resource or ephemeris variable is plotted as a point, that point represents the midpoint of the 'on' state and is calculated from the relation

$$t_{point} = t_{i1}+(t_{i2} -t_{i1})/2.0 \ .$$

For a task occurrence, however, the point represents the actual start-time of the task. In either case, the state is represented by a single data point (t_{point},X) where

$$X = HLIMIT(LINE,2) -HEIGHT/2.0 \ .$$

8.4.2 Plotting of Vertical Graphs

The flow diagram for the portion of Subroutine VPLOT concerned with placing the plot instructions on the IOPLT file is shown in Figure 8.7b. Like the horizontal plots, the vertical plots must be generated frame-by-frame and therefore the entire set of plot instructions must be repeated for every frame. Again, the exceptions are the calculations and instructions for grid backgrounds, labeling information, and scaling which do not change and therefore need to be specified only once.

The scale marks and accompanying units that are inserted along the Y axis are the same for each frame. Hence, though instructions must be issued once each frame to write them on the frame, the calculations to select the proper scale are performed only once before the plot instruction cycle begins. The program selects scale marks at the endpoints of the display area and at regular intervals in between. The scale units (i.e., days, hours, or minutes) are selected on the basis of the size of RPL. The program then selects the maximum number of grid intervals (and hence the number of interim scale marks) for which the remainder of

$$\frac{r_L}{un}$$

where

r_L - range per line (FORTRAN variable RPL)

u - scale units

n - number of intervals

is zero above two decimal places. A maximum of six intervals is permitted and n will be set to six if this requirement cannot be met.

The instruction cycle for the vertical plots is very similar to its horizontal counterpart (Figure 8.3b). Plotting instructions are again generated by successive calls to Subroutine QXY which specify that all data points between the lower and upper data limits (TL and TU respectively) are plotted on the display area. In order to produce the desired format, these factors must be redefined for each time interval rather than for each frame. Hence, in the basic instruction cycle, TL and TU are incremented by RPL, the scaling factors are redefined, and plot instructions are issued for each variable.

The value of the time interval counter LINE is incremented by one whenever TL and TU are redefined, and it is reset to one whenever its value exceeds the designated number of lines per frame (LPF). Labeling instructions for both the horizontal and vertical axes are issued whenever LINE is equal to one.

Each call to Subroutine QXY results in a call to Subroutine PLTXYQ in Phase 2 of AUPLOT. The subroutine will either plot the data as individual points or as a continuous curve depending upon the value of Flag IPCHAR which in turn is set by the call to Subroutine PLCHAR in Phase 1. If the value of IPCHAR is zero, the data will be plotted as a continuous curve. If the value of the flag is equal to I(I>0), the data is plotted as individual points using the Ith symbol in a table of symbols set aside for that purpose.

As noted above, Subroutine PLTXYQ was modified to permit the generation of shading instructions. However, a number of variables used in Subroutine VPLOT are needed to generate these instructions. The value of the required variables are transferred to Phase 2 by separate calls to Subroutine AUFLAG. The statement

CALL AUFLAG (I, VALUE)

places the current value of FORTRAN variable VALUE on the IOPLT file. When the instruction is read during the execution of Phase 2, this value will be placed in the Ith location of Array IEXTRA. The array contains 15 locations which are not used in the standard version of AUPLOT and hence can be used for special purposes through the Phase 2 COMMON structure. The values of all Phase 1 variables required by Subroutine PLTXYQ are input to the subroutine in this manner.

9.0 The ATS Job Decks

Each of the ATS programs is designed to run in the batch mode on the UNIVAC 1108 computer operating under the EXEC 8 multi-processing system. The user describes a run via a job deck, i.e., a single deck of punched cards containing all of the system instructions and input data needed to use a particular program. Sections 9.1 - 9.3 describe the job deck for each of the ATS programs. The descriptions assume a familiarity with the UNIVAC 1108 control statements and the NAMELIST input/output package. A complete description of the control statements can be found in Reference 13 and a description of the NAMELIST software package can be found in Reference 14.

9.1 Job Deck for the Data Bank Generator

A sample job deck for the ATS Data Bank Generator is shown in Table 9.1. Four files must be assigned for the duration of the run. The first is the program file ATS*SCHEDULER which is stored on FASTRAND and contains the Data Bank Generator absolute element BNKMAP. The file containing the Data Bank itself is designated by the user. It may be stored either on FASTRAND (the case shown) or on magnetic tape. In either case the file must be assigned to logical unit 2. Finally, two auxiliary storage files, TASKANNOT and TASKDESCR, are used to store the task annotation and description lists. These files must be stored on fast drum or FASTRAND to permit random access (Section 5.3.2) and must be assigned to logical units 1 and 3 respectively. Note that all files shown in Table 9.1 have the qualifier ATS as specified in the third field of the RUN card.

Only two variables are included in the NAMELIST statement, flags NEW and NOLIST. As shown in Figure 6.2, the value of NEW indicates whether an existing data bank is to be modified (NEW=0) or a completely new bank is to be created (NEW=1). Similarly, the value of NOLIST indicates whether the tasks in the bank are to be printed out as they are stored (NOLIST=0) or if that printing is to be suppressed (NOLIST=1). Since the values of both variables are initialized to zero at the start of the execution, they need only appear in the data deck if they are to be set equal to 1. The task description cards are placed after the NAMELIST data subject to the sequence rules discussed in Section 4.5.

Table 9.1

Job Deck for the ATS Data Bank Generator

```
@RUN        ABBJOB, ABB, ATS, 30, 200
@HDG        JOB DECK FOR ATS DATA BANK GENERATOR
@ASG,A      SCHEDULER
@ASG,T      TASKANNOT,F
@USE        1,TASKANNOT
@ASG,T      TSKDESCR,F
@USE        3,TASKDESCR
@ASG,A      DATABANK
@USE        2,DATABANK
@XQT        ATS*SCHEDULER•BNKMAP
 $INPUT
            [NAMELIST Variables]
 $END
            [Task Description Cards]
 LAST
@FIN
```

9.2 Job Deck for the Schedule Generator

A sample job deck for the ATS Schedule Generator is shown in Table 9.2. The structure of the deck is the same as that used for the Data Bank Generator. In this case however, the number of files assigned for the duration of the run depends upon the program options desired by the user. A minimum of three files must be assigned: the program file ATS*SCHEDULER containing the Schedule Generator absolute element SCDMAP and the two drum files, TASKANNOT and TASKDESCR, which will be used during the run to store the task Annotation and Descriptor Lists.

Each of the four remaining files need be assigned only if the option associated with that file is desired. If a data bank is to be used, the file containing the bank is assigned to logical unit 2. Similarly, if ephemeris information is to be input, the file containing that information is assigned to logical unit 4, or if the Schedule Generator is to be initialized from a History Tape, that tape is assigned to logical unit 8. Finally if a new History Tape is to be made, the output tape is assigned to logical unit 9. Although all of these optional files are shown as tape files in Table 9.2, any or all may be assigned as FASTRAND files.

The NAMELIST statement for the Schedule Generator contains the names of the 14 variables defined in Table 9.3. A subset of these variables must be specified for each run but the variables included in that subset depend upon the method of initialization and the program options to be exercised. Table 9.4 illustrates all of the data options available when a completely new schedule is to be generated while Table 9.5 illustrates the available options when a partial schedule is to be completed. As mentioned above, a variable need only be specified in an input data deck when the desired value is something other than zero.

9.2.1 Input Data to Generate a Completely New Schedule

When a completely new schedule is to be generated, the specification of all of the variables shown in Table 9.4 is optional with the exception of the variable TOTIME. In two cases, however, the ephemeris data option and the Data Bank option, the user must select an option from a subgroup of alternatives.

Table 9.2

Job Deck for the ATS Schedule Generator

```
@RUN        ATSJOB, ABB, ATS, 30, 200
@HDG        JOB DECK FOR THE ATS SCHEDULE GENERATOR
@ASG,A      SCHEDULER
@ASG,T      TASKANNOT.,F
@USE        1,TASKANNOT
@ASG,T      TASKDESCR.,F
@USE        3,TASKDESCR
@ASG,TM     2,T,xxxx      (assign Data Bank tape and relate
                           it to logical unit 2)*
@ASG,TM     4,T,xxxx      (assign ephemeris tape and relate
                           it to logical unit 4)*
@ASG,TM     8,T,xxxx      (assign input History Tape and
                           relate it to logical unit 8)*
@ASG,TM     9,T,xxxxR     (assign output History Tape and
                           relate it to logical unit 9)*
@XQT        ATS*SCHEDULER.SCDMAP
 $INPUT
          [NAMELIST Variables]
 $END
          [Task Description Cards]
 LAST
@FIN
```

*Note: xxxx denotes the reel number of the particular magnetic tape.

Table 9.3

Variables Included in the Schedule Generator NAMELIST Statement

FORTRAN Variable	Dimension		Data Format	Definition
GDATE	(3)	(1) (2) (3)	Hollerith Integer Integer	Gregorian date of launch as recorded on the ephemeris data file.*
IEPHEM	(1)		Integer	Ephemeris data flag, 0: included, 1: not included.
IGNORE	(200)		Hollerith	Array containing the names of the tasks whose descriptions are not to be transferred from the Permanent Data Bank.
INCLUD	(200)		Hollerith	Array containing the names of the tasks whose descriptions are to be obtained from the Permanent Data Bank.
IPRINT	(1)		Integer	Print Frequency Flag.
IPRIOR	(1)		Integer	Priority Level Indicator.
ITABLE	(1)		Integer	Start-time window table flag, 0:NO, 1:YES.
ITAPE	(1)		Integer	History Tape output flag, 0:YES, 1:NO.
NCRENT	(1)		Integer	Maximum number of entries in each crew commitment table.
NEWCOM	(3,20)	(1,X) (2,X) (3,X)	Hollerith Integer Integer	Array containing the name, type, and maximum # of permissible entries of every required resource commitment table except those for crewmen (maximum of 17).
NEWCRW	(2,5)		Hollerith	Array containing the name and associated skill of each crewman (maximum of 5).
NEWDAT	(3,10)	(1,X) (2,X) (3,X)	Hollerith Floating Point Floating Point	Array containing the name, initial quantity, and maximum usage rate of each consumable (maximum of 10).
TIMEL	(1)		Floating Point	KSC time of launch as recorded on the ephemeris data file (hours)
TOTIME	(1)		Floating Point	Total mission time (days).

*The date is entered as a three-tuple. The first element contains one of the standard three letter codes for the month, the second element contains a one or two digit integer specifying the day of the month, and the third element contains a four digit integer specifying the year. An example is shown in Table 9.4.

Table 9.4

NAMELIST Data Specifications for the Generation of a New Schedule

Specify Total Mission Time	
TOTIME = 26.0,	Specify the mission duration as 26.0 days.
Ephemeris Options (Specify one)	
a. IEPHEM=1,	An ephemeris tape will not be used.
or	
b. GDATE='JUL', 16, 1972, TIMEL=14.583,	The values of GDATE and TIMEL that are identical with the date and KSC time of launch contained on the ATS ephemeris tape.
Data Bank Options (Specify one)	
a. INCLUD = 'TASKA', 'TASKB', 'TASKC', ...	Descriptions of the specified tasks will be copied from the permanent Data Bank.
or	
b. INCLUD = 'ALL',	Descriptions of all tasks in the permanent Data Bank will be copied.
or	
c. IGNORE = 'TASKW', 'TASKX', 'TASKY', ...	Descriptions of all tasks except those specified will be copied from the permanent Data Bank.
or	
d. IGNORE = 'ALL',	No permanent Data Bank will be used for this run.
History Tape Option	
ITAPE=1,	No output History Tape will be generated.

Table 9.4 (cont'd)

Define Resource Commitment Tables

NEWCOM = 'POWER', 150, 2,	A resource commitment table for an analog resource named POWER is to be established with a maximum of 150 entries.
NEWCRW = 'CREWD', 'NONE', 'CREWC', 'DOCTOR',	Resource commitment tables for crewmen CREWC and CREWD are to be established. Crewman CREWD will have no specific skill; crewman CREWC will be assigned the skill of DOCTOR.
NCRENT = 300,	Resource commitment tables for all crewmen are permitted a maximum of 300 entries.

Define Consumable Limits

NEWDAT = 'OXYGEN', 5000.0, 0.0, 'POWER', 0.0, 4000.0,	Two consumables named OXYGEN and POWER are to be tracked. OXYGEN has an initial amount of 5000.0 available and an unspecified usage rate (designated by 0.0). POWER has a unspecified amount available (designated by 0.0) and a maximum usage rage of 4000.0.

Print Options

IPRINT=4,	All of the resource commitment tables are to be printed out every four priority levels.
ITABLE=1,	A table showing the duration of start-time windows is to be printed.

Table 9.5

NAMELIST Data Specifications to Complete a Partial Schedule

Specify Priority Level	
IPRIOR = 10,	Read appropriate record from the input History Tape to initialize all tables to their status after all of the tasks at the tenth priority level have been considered. The scheduling process begins with tasks having a priority equal to 11.
History Tape Option	
ITAPE = 1,	No output History Tape will be generated.
Print Options	
IPRINT = 4	All of the resource commitment tables are to be printed out every four priority levels.
ITABLE = 1	A table showing the duration of start-time windows is to be printed.

9.2.1.1 Ephemeris Data Option

If no ephemeris data is to be input, the flag IEPHEM is set equal to one. Conversely, if a file containing ephemeris data is to be input, the variables GDATE and TIMEL must be specified exactly as they appear on that file. If either or both of these variables do not match their counterparts on the ephemeris data file, the run will terminate in an error message (Appendix A - Section A.2.8.1).

9.2.1.2 Data Bank Option

Four data bank options are provided to minimize the number of individual task names that have to be specified in the data deck. Normally, array INCLUD contains the names of all of the tasks whose descriptions are to be copied from the permanent Data Bank. If all of the descriptions in the bank are to be used, the user may set INCLUD equal to 'ALL' rather than specify each task name separately. If all but a few tasks are to be used, the user may alternately specify the names of those tasks to be omitted in the IGNORE array. If no Data Bank is to be used, the variable IGNORE is set equal to 'ALL'.

9.2.1.3 History Tape Option

Only one option is available. If no output history tape is to be generated, the variable ITAPE is set equal to one.

9.2.1.4 Specification of Resource Commitment Tables

As noted in Section 5.1, the characteristics for each resource commitment table must be supplied as input data. Three characteristics of each resource commitment table must be supplied: the name, the maximum number of entries in the table, and the number of columns in the table (two for analog resources, three for binary resources). The characteristics for each table are entered as successive three-tuples* in the one-dimensional array NEWCOM. Thus, for the example shown in Table 9.4, a resource commitment table named POWER is defined as an analog table (two columns) with a maximum of 150 entries.

---

*An n-tuple is an ordered set of n elements.

All resource commitment tables except those for crewmen are specified through array NEWCOM.* The two characteristics of each crewman, name and skill, are entered as successive two-tuples in the one-dimensional array NEWCRW. For convenience, the program already contains the names of three crewmen: CREWA, CREWB, and CREWC. The corresponding resource commitment tables have each been allotted 400 entries. If additional crewmen are required, their names must be specified in array NEWCRW along with a designated skill (one skill per crewman). If no specific skill is to be assigned to the crewman, the word 'NONE' should be entered. The three crewmen already designated (CREWA, CREWB, CREWC) have no specific skills assigned. To assign a skill to one of these crewmen, the name and skill are entered in array NEWCRW as if it were a new entry. Array NEWCRW will accept a maximum of five two-tuples.

In the example shown in Table 9.4, Array NEWCRW will have two entries. First, a new crewman named CREWD is defined without a specific skill. In addition, crewman CREWC, already defined by the program, is assigned the skill DOCTOR. The resource commitment table for each crewman is allocated a maximum of 400 entries by the program. If this is unsatisfactory, the user may assign a new maximum via variable NCRENT.

The user is cautioned on two points. The first is to make realistic estimates of the maximum number of entries for each resource commitment table. As explained in Section 5.1, these maximums are used to allocate space in the WA Array. The number of locations allocated to each table in the array is the product of the maximum number of entries (the actual number rather than the estimated maximum is used for ephemeris resource tables) and the number of columns in the table. If the total number of locations (i.e., the sum of the products) required exceeds the dimension of the WA Array, the run will be terminated with an error message (Appendix A - Section A.2.8.2). The WA Array is currently dimensioned to 12,000 locations.

The second point is that Array LTABLE, which acts as the table of contents to the WA Array (Section 5.1) is currently dimensioned to 20. Therefore, no more than 20 commitment tables (resource and ephemeris) may be specified.

---

*Ephemeris resource tables are defined directly from the ephemeris data tape and therefore need not be specified in the input data deck.

9.2.1.5 Specification of Consumables Limits

As noted in Section 7.2, the maximum quantities and/or usage rates for each consumable must be supplied as input data. Three characteristics, name, initial quantity available, and maximum usage rate, for each table are entered as successive three-tuples in the one-dimensional Array NEWDAT. If either of the qualitative characteristics do not apply to the particular consumable, a value of zero should be entered. The example in Table 9.4 shows that two consumables, named OXYGEN and POWER, are entered into Array NEWDAT. OXYGEN has an initial quantity of 5000.0 available and no specified usage rate. The consumable POWER has no specified quantity available but a maximum usage rate of 4000.0. Note that no units are specified for either the amount available or the usage rate. To obtain meaningful results, the user must insure that the quantities specified for the initial amount available are consistent with the quantities specified for that consumable on AMOUNT cards. Similarly, the user must also insure that the maximum usage rate specified in NEWDAT is consistent with the usage rates specified in the sixth field on the Resource Cards.

9.2.1.6 Print Options

Two print options are available. The Print Frequency Flag IPRINT is used to control the printing of intermediate results. When the flag is set equal to k, the program will print out the contents of all resource commitment tables every k priority levels. If IPRINT is not specified, only the final results will be printed.

The variable ITABLE is used to control the printing of a table that illustrates the calculation of start-time windows for each task. This option is usually used only for detailed analysis of particular problems since it results in a large volume of printout.

9.2.2 Input Data to Complete a Partial Schedule

Comparison of Tables 9.4 and 9.5 shows that the available input options are significantly narrowed when a partial schedule is to be completed. The variables for the two print options (IPRINT and ITABLE) as well as the option for an output History Tape are the same as described in Section 9.2.1. In addition, the value for the priority

level indicator IPRIOR must be specified. When IPRIOR is set equal to k, the schedule will be initialized at the end of the kth priority level (Section 7.2).

9.2.3 Sample Job Decks

Job decks to illustrate each of the basic options are shown in Tables 9.6 and 9.7. Table 9.6 illustrates a job deck to generate a new schedule. A Data Bank stored on magnetic tape #1000 is assigned to logical unit 2 while an ephemeris data file placed on tape #1100 is assigned to logical unit 4. An input History Tape is to be generated and stored on tape #1200.

The ephemeris and data bank options in the data deck are consistent with the file assignments. In addition, two resource commitment tables are defined. The first is for an analog resource named POWER which may contain up to 175 entries while the second is for a binary resource named SCILOK. The latter may contain a maximum of 200 entries. The maximum usage rate for the analog resource POWER is defined as 4000.0. The deck of task description cards (Section 4.5) following the input NAMELIST contains edits to task descriptions for Task SLEEP and defines a new task named TASKX. Note that the description of TASKX will be stored on the output History Tape as well as on the auxiliary drum files TASKANNOT and TASKDESCR, thus making that task description available to any subsequent runs using that History Tape (tape #1200) as an input.

Table 9.7 illustrates a job deck that could be used to complete a partial schedule. Tape #1200, the History Tape generated from the run described in Table 9.6 is used as the input History Tape for this run and is therefore assigned to logical unit 8. The data deck indicates that the schedule is to be initialized at the end of priority level 12, e.g., after all of the entries at the 12th priority level have been made. In this case, the deck of task description cards following the NAMELIST input consists of only one edit. The card redefines the priority of task TASKX so that it will be considered at the 13th priority level rather than the 20th as it was in the previous schedule.

9.3 Job Deck for the Data Processor

The general structure of the job deck for the ATS Data Processor is shown in Table 9.8. All four files shown in the table must be assigned for every run. The program file ATS*PROCESSOR contains the absolute elements PLTMAP and

Table 9.6

Sample Job Deck for the Generation of a New Schedule

```
@RUN          ABBNEW, ABB, ATS, 30, 200
@HDG          JOB DECK TO GENERATE A NEW SCHEDULE
@ASG,A        SCHEDULER
@ASG,T        TASKANNOT.,F
@USE          1,TASKANNOT
@ASG,T        TASKDESCR.,F
@USE          3,TASKDESCR
@ASG,TM       2,T,1000
@ASG,TM       4,T,1100
@ASG,TM       9,T,1200R
@XQT          ATS*SCHEDULER·SCDMAP
 $INPUT
 GDATE =  'JUL', 16, 1972,
 TIMEL =  14.583
 IGNORE=  'M093A',
 NEWCOM=  'POWER', 175, 2, 'SCILOK, 200, 3
 NEWDAT=  'POWER', 0.0, 4000.0,
 IPRINT=  5,
 $END
SLEEP/COMMNT,         4,     TO 00:10:00 MISSION ELAPSED TIME
SLEEP/TIME,           00:10:00
TASKX/TITLE,          1      Task TASKX
TASKX/COMMNT,         1      THIS TASK REQUIRES THE SERVICES OF
TASKX/COMMNT,         2      CREWMAN CREWA FOR 30 MINUTES.
TASKX/PRI,            20
TASKX/OBJEC,          1,1
TASKX/RES,            CREWA,   00:00:00, 00:00:30
LAST
@FIN
```

Table 9.7

Sample Job Deck to Complete a Partial Schedule

```
@RUN          ABBPAR, ABB, ATS, 30, 200
@HDG          JOB DECK TO COMPLETE A PARTIAL SCHEDULE
@ASG,A        ATS*SCHEDULER
@ASG,T        TASKANNOT.,F
@USE          1,TASKANNOT
@ASG,T        TASKDESCR.,F
@USE          3,TASKDESCR
@ASG,TM       8,T,1200
@XQT          ATS*SCHEDULER·SCDMAP
 $INPUT
 IPRIOR=12,
 ITAPE=1,
 IPRINT=1,
 $END
TASKX/PRI,  13
LAST
@FIN
```

Table 9.8

Structure of the Job Deck for the ATS Data Processor

```
@RUN        ABBPLT, ABB, ATS, 30, 200
@HDG        JOB DECK FOR THE ATS DATA PROCESSOR
@ASG,A      PROCESSOR
@ASG,T      TASKANNOT.,F
@USE        1,TASKANNOT
@ASG,T      TASKDESCR.,F
@USE        3,TASKDESCR
@ASG,TM     8,T,xxxx        (Assign input History Tape and relate
                             it to the logical unit 8)*
@ASG,TM     PLOTFILE., T,PLOT     (Assign output plot tape)
@XQT        ATS*PROCESSOR·PLTMAP
 $INPUT
                [NAMELIST Variables]
 $END
 $INPUT
                [NAMELIST Variables]
 $END
@XQT        ATS*PROCESSOR·PHASE2
@FIN
```

---

*Note: xxxx denotes the reel number of the particular magnetic tape.

PHASE2 which must be executed in the sequence shown to produce the output plot tape that is input to the SC-4020 plotter. The drum files TASKANNOT and TASKDESCR again contain the task annotation and descriptor lists which are obtained from the input History Tape.

As noted in Section 8.2, any number of plots may be generated in a single run but each set of plot description data must be input separately. Each plot is described in a separate set of NAMELIST input data and so more than one set may be included in a job deck.

9.3.1 Processor Data Deck

The NAMELIST statement for the Data Processor contains the 15 variables defined in Table 9.9. A subset of these variables must be specified in each set of NAMELIST input data but the variables included in that set depend upon the type of graph requested. Table 9.10 lists the specification options available to the user. Note that the variables marked with an asterisk will retain their values for the duration of the run. Once specified, therefore, these variables need only be redefined when the values must be changed.

9.3.1.1 Priority Level Option

The input value of IPRIOR serves the same function in the Data Processor as it does in the Schedule Generator: to provide scheduling data representing a particular point in the scheduling process. A value of IPRIOR>0 must be specified.

9.3.1.2 Plot Interval

TBEGIN and TEND, the lower and upper endpoints of the plot interval must be specified for every plot.

9.3.1.3 Graph Label

Array TITLE contains the 48 characters that will be written at the top of each frame of the graph. Any title containing a maximum of 48 characters (including blanks) may be specified. The title will be printed exactly as it appears between the quote marks on the input card.

9.3.1.4 Specifications for Coaxial (Horizontal) Plots

Only two variables must be specified to describe a horizontal plot. HDEP contains the names of the variables

Table 9.9

Variables Included in the Data Processor NAMELIST Statement

FORTRAN Variable	Dimension	Data Format	Definition
HDEP	(5)	Hollerith	Array containing the names of the variables to be included on a coaxial (horizontal) graph (maximum of 5).
IPRIOR	(1)	Integer	Priority Level Indicator
KDUMP	(1)	Integer	Flag to control the printing out of all collected data. 0:NO, 1:YES.
LAST	(1)	Integer	Flag indicating the last plot request.
LPF	(1)	Integer	Number of time intervals (lines) to be placed on one frame.
RPF	(1)	Floating Point	Length of the time interval to be displayed on one frame (days).
RPL	(1)	Floating Point	Length of one time interval (days).
TBEGIN	(1)	Floating Point	Lower endpoint of the plot interval (days).
TEND	(1)	Floating Point	Upper endpoint of the plot interval (days).
TITLE	(8)	Hollerith	Array containing the 48 alphanumeric characters to be used as the plot title.
VGRID	(1)	Integer	Grid Indicator Flag for periodic plots: 0:YES, 1:NO.
VSHADE	(3)	Hollerith	Array containing the names of the variables whose occurrences are to be represented as shaded boxes on a period plot (maximum of 3).
VPOINT	(15)	Hollerith	Array containing the names of the variables whose occurrences are to be represented as points on a periodic plot (maximum of 15).

Table 9.10

NAMELIST SPECIFICATION OPTIONS FOR THE ATS DATA PROCESSOR

Priority Level	
* IPRIOR = 10,	Read appropriate record from the input History Tape to initialize all tables to their status after all of the tasks at the 10th priority level have been considered.
Plot Interval	
TBEGIN = 1.0,	All points between 1.0 and 2.0 days are to be plotted.
TEND = 2.0,	
Graph Label	
* TITLE = 'GRAPH TITLE WITH A MAXIMUM OF 48 CHARACTERS',	The title 'GRAPH TITLE WITH A MAXIMUM OF 48 CHARACTERS' is to be printed out across the top of every frame of the graph.
Coaxial Plot Variables	
HDEP = 'CREWA', 'TASKX', 'POWER', . . .	Data for the variables named CREWA, TASKX, and POWER are to be plotted on a coaxial graph.
RPF = 1.0,	The coaxial graph is to be plotted to a horizontal scale of 1.0 days per frame.

* Value is retained until redefined.

Table 9.10 (Cont.)

	Periodic Plot Variables	
	VPOINT = 'TASKX', . . .	Occurrences of the variable TASKX are to be plotted as points in a periodic plot.
	VSHADE = 'CREWA', . . .	Occurrences of the variable CREWA are to be plotted as shaded boxes on a periodic plot.
	RPL = 2.0,	One time interval (line) on a periodic plot is to represent 2.0 days.
*	LPF = 10,	10 time intervals (lines) are to appear on one frame of a periodic plot.
	VGRID = 1	Grid lines on the periodic plot are to be suppressed.
	Program Control	
*	KDUMP = 1,	All collected data is to be printed out.
	LAST = 1,	This is the last plot request.

* Value is retained until redefined.

to be plotted while the value of variable RPF specifies the length of the time interval to be displayed on one frame.

9.3.1.5 Specifications for Periodic Plots

As noted in Section 8.2, the user must specify in the input data how the occurrences of each variable are to be displayed. Array VPOINT therefore contains the names of the variables whose occurrences are to be displayed as points while Array VSHADE contains the names of those variables whose occurrences are to be displayed as shaded boxes. RPL and LPF must also be specified. Finally, the flag VGRID is set equal to one if the grid lines on the plot are to be suppressed.

9.3.1.6 Program Control Options

The variable KDUMP controls the printout of collected data. When KDUMP is set equal to one in the data deck all of the data points for each variable will be printed out during the execution of Phase 2. This option is used primarily as a diagnostic tool. Finally, the variable LAST must be set equal to one in the last set of plot data in order to terminate the execution of Phase 1.

9.3.2 Sample Job Deck

Table 9.11 shows the job deck required to generate the three graphs illustrated in Figures 3.2 and 3.3. Three sets of NAMELIST input data are required. The first set requests that the entries in the resource commitment tables for each of the three crewmen be displayed on a coaxial plot. All entries between two and three days are to be displayed to a scale of one day per frame. The setting of the variable IPRIOR requires that the data at the 44th priority level be displayed.

The second data set also requests coaxial plots. The third set, however, requests periodic plots. In the latter set, the occurrences of the Task LUNCHA are to be plotted as points while the occurrences of the variable DAY and the Task REST are to be displayed as shaded boxes. All occurrences between one and seven days are to be displayed to scales of one day per line and six lines per frame. Since IPRIOR is not specified in the second or third data sets, the corresponding graphs will also use the data from the 44th priority level.

Table 9.11

Sample Job Deck for the ATS Data Processor

```
@RUN       ABBPLT, ABB, ATS, 30, 200
@HDG       JOB DECK TO GENERATE HORIZONTAL & VERTICAL PLOTS
@ASG,A     PROCESSOR
@ASG,T     TASKANNOT.,F
@USE,      1,TASKANNOT
@ASG,A     TASKDESCR.,F
@USE       3,TASKDESCR
@ASG,TM    8,T,1200
@ASG,TM    PLOTFILE.,T,PLOT
@XQT       ATS*PROCESSOR•PLTMAP
 $INPUT
 IPRIOR=44,
 TBEGIN=2.0,
 TEND=3.0,
 TITLE='COAXIAL PLOTS OF CREW TIMELINES',
 HDEP='CREWA', 'CREWB', 'CREWC',
 RPF=1.0,
 $END
 $INPUT
 TBEGIN=2.0
 TEND=3.0,
 TITLE='COAXIAL PLOTS OF CREWA, LUNCHA, POWER, AND S/C DAY',
 HDEP='CREWA', 'LUNCHA', 'POWER', 'DAY',
 RPF=1.0
 $END
```

Table 9.11 (cont'd)

```
 $INPUT
 TBEGIN=1.0,
 TEND=7.0,
 TITLE='PERIODIC PLOTS OF S/C DAY, REST, AND LUNCHA',
 VPOINT='LUNCHA',
 VSHADE='DAY', 'REST',
 RPL=1.0,
 LPF=6,
 LAST=1,
 $END
@XQT     ATS*PROCESSOR•PHASE2
@FIN
```

10.0 Recommendations for Future Work

The implementation of the ATS represents the first phase in an investigation into the nature of the scheduling process. Future work on the ATS should be concerned with expanding its capabilities to meet the operational needs arising from two areas of investigation. The first is concerned with defining one or more parameters to measure the effectiveness of a particular schedule, i.e., parameters that will aid the user in evaluating the relative merits of different schedules. Since the nature of these parameters cannot be completely anticipated, the feasibility of expanding the Data Processor to perform any set of user-formulated calculations will be investigated.

The second area of investigation should be directed toward determining a meaningful dynamic priority system which could be overlaid on the static priority system now in the ATS. The dynamic system would reflect the effect of current scheduling decisions on future opportunities and would indicate the order in which a subgroup of tasks having the same static priority rating should be scheduled. Allied with this problem is the selection of the proper start-time from within the defined start-time windows. In the present version, the earliest possible start-time is chosen arbitrarily but a selection made on a more analytical basis is obviously more desirable. The first step in this investigation will be to modify the Schedule Generator to keep track of the reason for tasks not being scheduled. This modification will enable the user to quickly determine which of the task requirements or constraints is preventing the scheduling of the task. This knowledge should lead to a broader understanding of the cause and effect relationship between current decisions and future alternatives.

11.0 Summary

The Automated Task Scheduler, a group of computer programs designed to produce and display mission timelines (or schedules) for manned space missions, has been implemented. The system consists of three computer programs:

1. A Schedule Generator that generates a time history of the commitments for each designated resource and a corresponding list of start-times for each task.

2. A Data Processor that displays timeline data in graphical form.

3. A Data Bank Generator that creates and edits a permanent task Data Bank.

The primary output of the Schedule Generator is a set of resource commitment tables and a list of start-times for each task. On option, however, intermediate results can be recorded on a History Tape which can be used on a subsequent run to initiate the scheduling process at some intermediate point. Hence, the Schedule Generator can be used to complete a partial schedule as well as generate a completely new schedule.

The History Tape is also used as an input to the Data Processor which is used for graphically displaying the timeline data produced by the Schedule Generator. Two types of plots can be generated: coaxial and periodic. Coaxial plots can display up to five binary and/or analog variables on a single set of axes. The periodic plots are used to overlay the occurrences of different binary variables and hence illustrate recurring patterns and the interrelationship between different variables.

One of the most important tasks in implementing the ATS was the development of a flexible input language capable of translating a wide variety of resource requirements and performance constraints into statements that could be recognized by the Schedule Generator. The resulting Task Description Language (TDL) consists of 12 card formats. Seven of these formats (the Priority, Objective, Time, Amount, Enable, Inhibit, and Resource Cards) are used to translate resource requirements, performance restrictions, and performance objectives into statements that can be

processed by the Schedule Generator. Two additional formats (the Title and Comment Cards) are provided to annotate the Descriptor Cards with alphanumeric comments. The final three formats (the Equivalence, Delete, and Last Cards) are used as control statements.

If the user establishes a data base consisting of a large number of task descriptions, he may, for convenience, store these descriptions on a Data Bank file using the Data Bank Generator Program. The descriptions on this bank can then be selectively copied by the Schedule Generator for each run, thus relieving the user of having to input a large number of cards. The task descriptions for the Schedule Generator run are stored on temporary drum files for the duration of that run and hence may be edited for that run without affecting the descriptions stored on the Permanent Data Bank.

When generating a schedule, the Schedule Generator considers each task once, the order of consideration being specified by the user. For each task, the program first determines acceptable start-time windows (i.e, continuous intervals during which the task may be initiated) and then selects start-times for as many repetitions of the task as are required by selecting points from within the windows. The process continues until all tasks have been considered. When a performance of a task is scheduled, the commitment tables for each resource required by the task are updated to reflect the commitment of that resource to that task.

Two methods of dynamic storage allocation were used in implementing the ATS algorithm in order to reduce the amount of required core space. The ephemeris and resource tables are all stored in one large working array that is partitioned off at the start of the program according to user estimates of table size. In contrast, the task descriptions, start-time windows, and lists of task start-times are stored in a list structure using SAC-1, a FORTRAN-imbedded list-processing language.

Acknowledgment

The author wishes to express his appreciation to Miss M. P. Odle and to Mr. R. F. Jessup for their advice on the use of the SAC-1 and AUPLOT Systems respectively, and to Miss D. P. Nash who helped with the programming of the ATS.

1025-ABB-1i A. B. Baker

Attachments

References

1. A. B. Baker
"A Survey of Automated Scheduling Models"
Bellcomm Memorandum for File B69 04020, April 7, 1969.

2. D. P. Nash
"ATSEPHEM - A Program to Generate an Ephemeris Data Tape for the Automated Task Scheduler"
Bellcomm Memorandum for File In Preparation

3. M. S. Feldman
"'Pick a Day" Experiment M093 Requirements - Case 610"
Bellcomm Memorandum for File B70 04026, April 7, 1970.

4. Ivan Flores, Computer Programming, Prentice-Hall, Inc., Englewood Cliffs, N. J. (1966)
Chapter 9, pp. 257-262.

5. Knuth, Donald E., The Art of Computer Programming, Vol. I, Addison-Wesley, Reading, Mass. (1968)
Chapter 2, pp. 228-435.

6. Sherman, P. M., Techniques in Computer Programming, Prentice-Hall, Inc., Englewood Cliffs, N. J. (1970)
Chapter 12, pp. 226-233.

7. G. E. Collins
"The SAC-1 List Processing System"
Computer Sciences Department
University of Wisconsin, July 11, 1967.

8. M. P. Odle
"SAC-1 System for List Processing and Formula Manipulation"
Bellcomm Memorandum for File B70 04001, April 1, 1970.

9. D. P. Nash
"Processing Task Description Cards in the Automated Task Scheduler"
Bellcomm Memorandum for File B70 12062, December 23, 1970.

10. "Programmers' Reference Manual
SC-4020 High-Speed Microfilm Recorder"
Stromberg-Carlson Document #9500056, October 1964.

References (cont'd)

11. R. F. Jessup
"AUPLOT System Description"
Bellcomm Memorandum for File B69 10116 October 27, 1969

12. R. F. Jessup
"AUPLOT II - A System of Data Handling and Plot Subroutines for Computer Graphics"
TM-70-2011-2 November 20, 1970

13. "EXEC 8 - User's Manual"
Bellcomm, Inc. August 1968
Section V

14. "UNIVAC 1108 Multi-Processor System Programmer's Reference Manual"
Sperry-Rand Corporation, 1969
pp. 6-13 through 6-17.

APPENDIX

ATS Error Diagnostics

A.0 Introduction

The wide variations in input data permitted by each of the ATS programs make it almost inevitable that the user will, at some time, violate one or more of the usage rules described in this manual. These violations will usually result in either the generation of erroneous information or the termination of the run by the computer executive. In order to prevent these alternatives and to aid the user in pinpointing these violations, a system of error terminations has been incorporated into each of the ATS programs. A diagnostic message is printed out with every termination. Each message includes the name of the subroutine in which the message is generated and the reason for the termination.

The messages generated by each of the ATS programs are presented in Sections A.1 through A.3. Within each section the messages are grouped according to the subroutines in which they are generated. Each message is printed in capital letters and is accompanied by an explanation. In all cases, the lower case letter k and names appearing inside quotation marks are inserted for purposes of illustration. They will be replaced with real data in the computer printout.

A.1 Data Bank Generator Diagnostic Messages

A.1.1 THE FIRST DESCRIPTOR CARD FOR TASK 'TASKX' IS NOT A PRIORITY CARD

The first Descriptor Card encountered for any task must be a Priority Card. The user should rearrange the input sequence of Task Description Cards so that the sequence rules described in Section 4.3.1 are satisfied.

A.1.2 THE PERFORMANCE OBJECTIVES FOR TASK 'TASKX' HAVE NOT BEEN DEFINED

This message is generated when the second Descriptor Card encountered for Task TASKX is not an Objective Card. The user should rearrange the input sequence of Task Description Cards so that the sequence rules described in Section 4.3.1 are satisfied.

A.1.3 THE CARD TYPE 'TYPEX' SPECIFIED IN FIELD 2 OF A DESCRIPTOR CARD FOR TASK 'TASKX' IS NOT A VALID CARD TYPE

The alphanumeric combination 'TYPEX', specified in Field 2 of a Task Description Card, does not match any of the 11 card types described in Section 4.2. The user should insure that the designation in Field 2 matches one of the type designators shown in Table 4.2 and that all of the field delimiters have been included.

A.1.4 TASK 'TASKY' NAMED ON FIELD 3 OF THE EQUIVALENCE CARD FOR TASK 'TASKX' HAS NOT BEEN DEFINED

This message will be generated by either of two possible errors. Either the name of the task appearing in Field 3 does not appear exactly as it appeared on its own

set of Task Description Cards or the input sequence of Task Description Cards is incorrect. In the latter case, the sequence must be rearranged so that the task named in Field 3 is defined before the Equivalence Card is encountered (Section 4.3.2).

A.1.5 THE FIRST DESCRIPTION CARD FOR TASK 'TASKX' IS NOT A TITLE CARD

The first Description Card for any task must be a Title Card. The user should rearrange the input of Task Description Cards so that the sequence rules described in Section 4.3.1 are satisfied.

A.1.6 THE NUMBER OF TASKS TO BE STORED IN THE DATA BANK, k, IS GREATER THAN THE NUMBER OF ENTRIES PERMITTED IN THE TOC ARRAY

The number of tasks that can be stored in the Data Bank is limited by the dimension of the TOC Array (Section 6.0). This message indicates that the maximum has been exceeded.

A.1.7 OUT OF AVAILABLE SPACE

This message is generated by Subroutine NOAVLS, an element in the SAC-1 System. The message indicates that there are no cells cn the Available Space List (Section 5.0). Since cells are continually being returned to the list, the complete deletion of the list could arise from a unique combination of schedule characteristics and data generation. A slight variation in these characteristics (e.g. varying the order in which the tasks are considered) usually eliminates the problem. If the problem persists, the Available Space List should be increased by increasing the dimension of Array ASL.

A.2 Schedule Generator Diagnostic Messages

A.2.1 Messages Generated in Subroutine CREW

A.2.1.1

TASK 'TASKX' HAS A MULTIPLE
REQUIREMENT ON CREWMAN 'CREWX'
BY SPECIFIC NAME AND BY SKILL
'SKILLX'

The task description for TASKX contains a Resource Card that specifies Crewman CREWX in Field 3 and another Resource Card that specifies skill SKILLX in Field 3. Since Crewman CREWX has been assigned SKILLX, the effect is to have more than one requirement on the same resource in the same task description which is not permitted (Section 4.4).

A.2.1.2

THE NUMBER OF UNDESIGNATED
CREWMEN FOR 'TASKX' EXCEEDS
THE NUMBER OF UNASSIGNED
CREWMEN AVAILABLE

The number of Resource Cards for Task TASKX with the ANY designation in Field 3 exceeds the number of unassigned crewmen (i.e., the total number of crewmen less the number designated by specific name and skill) available.

A.2.2 Message Generated in Subroutine ENTRY

A.2.2.1

SCHEDULING OF TASK 'TASKX' WILL
EXCEED THE ALLOTTED SPACE IN
THE 'RESNME' COMMITMENT TABLE

If the RESNME commitment table is updated to reflect the allocation of that resource to Task TASKX, the maximum number of entries permitted for that table will be exceeded (Section 5.1.2).

A.2.3 Messages Generated in Routine MAIN

A.2.3.1 THE RESOURCE 'RESNME' NAME ON FIELD 3 OF AN AMOUNT CARD FOR TASK 'TASKX' DOES NOT APPEAR IN ARRAY DTABLE

The alphanumeric combination RESNME does not match any of the names specified as consumables in Array DTABLE. The user should check to insure that the name specified on the Amount Card is identical to the name specified in the NAMELIST input via Array NEWDAT (Sections 4.2.1.7 and 9.2.1).

A.2.3.2 THE NUMBER OF TASKS, k, TO BE SCHEDULED IN THIS RUN IS GREATER THAN THE NUMBER OF ENTRIES PERMITTED IN THE LNAMES AND ORDER ARRAYS

The maximum number of tasks that can be scheduled in any one run of the Schedule Generator is limited by the dimension of the LNAMES and ORDER Arrays. This message indicates that the maximum has been exceeded. The current version of the ATS permits a maximum of 200 tasks to be considered for scheduling in any one run. This maximum can only be increased by increasing the dimension of the LNAMES and ORDER Arrays (Section 7.0).

A.2.3.3 THE INDEPENDENT TASK 'TASKX', NAMED ON A 'TYPE' CARD FOR DEPENDENT TASK 'TASKY', DOES NOT APPEAR IN THE ORDER ARRAY

This message indicates that the dependent task, TASKY, has become a candidate for scheduling before the independent task named on an Enable or Inhibit Card which is not permitted (Sections 4.2.1.5 and 4.2.1.6).

A.2.3.4 OUT OF AVAILABLE SPACE

See Section A.1.7.

A.2.4 Message Generated in Subroutine MULTI

A.2.4.1 THE NUMBER OF WINDOWS IN THE k SECTION OF START-TIME WINDOWS FOR TASK 'TASKX' IS GREATER THAN THE MAXIMUM SPACE ALLOTTED IN ARRAY WIN

The amount of data to be entered into the internal working Array WIN exceeds the maximum dimensions of that array. The amount of data (in this case, lists of start-time windows) generated depends upon a unique combination of time-line characteristics that exist at the point in the scheduling process where TASKX is being considered. A slight variation in these commitments (e.g., in the order in which the tasks are considered for scheduling) will usually eliminate this problem. If the problem reoccurs the dimension of Array WIN should be increased.

A.2.5 Messages Generated by Subroutine RESRCE

A.2.5.1 THE NAME 'RESNME' SPECIFIED IN FIELD 3 OF A RESOURCE CARD FOR TASK 'TASKX' DOES NOT MATCH ANY NAME IN ARRAY LTABLE

The alphanumeric combination 'RESNME' does not match any of the names of the commitment tables stored in Array LTABLE (Section 5.1). The user should insure that the name specified on the Resource Card is identical to one of the names specified in the input data via Arrays NEWCOM and NEWCRW in the NAMELIST input or via the ATS Ephemeris Tape. This message will also be generated if the resource name defined in the input data ends on a numeral rather than a letter as required (Sections 4.2.1.7 and 9.2.1).

A.2.5.2 THE NAME 'RESNME' SPECIFIED IN FIELD 3 OF A RESOURCE CARD FOR TASK 'TASKX' DOES NOT MATCH ANY NAME IN ARRAY DTABLE

The alphanumeric combination 'RESNME' does not match any of the names specified as consumables in Array DTABLE. The user should check to insure that the name specified on the Resource Card is identical to the name specified in the NAMELIST input via Array NEWDAT (Section 9.2.1).

A.2.6 Messages Generated in Subroutine SCHED

A.2.6.1 THE NUMBER OF SECTIONS OF START-TIME WINDOWS FOR TASK 'TASKX' IS GREATER THAN THE NUMBER OF ENTRIES PERMITTED IN THE SEC ARRAY

The amount of data to be entered into the internal working Array SEC exceeds the maximum dimensions of that array. The amount of data (in this case, lists of start-time windows) generated depends upon a unique combination of time-line characteristics that exist at the point in the scheduling process where TASKX is being considered. A slight variation in these commitments (e.g., on the order in which the tasks are considered for scheduling) will usually eliminate this problem. If the problem reoccurs, the dimension of Array SEC should be increased.

A.2.6.2 THE DESIRED NUMBER OF PERFORMANCES, k, SPECIFIED FOR TASK 'TASKX' EXCEEDS THE NUMBER OF ENTRIES PERMITTED IN ARRAY LVWIN

The desired number of performances specified in Field 4 of the Objective Card for Task TASKX exceeds the row dimension of Array LVWIN (Section 4.2.1.2).

A.2.7 Messages Generated by Subroutine SETUP

A.2.7.1 THE FIRST DESCRIPTOR CARD FOR TASK 'TASKX' IS NOT A PRIORITY CARD

See Section A.1.1.

A.2.7.2 THE PERFORMANCE OBJECTIVES FOR TASK 'TASKX' HAVE NOT BEEN DEFINED

See Section A.1.2.

A.2.7.3 THE CARD TYPE 'TYPEX' SPECIFIED ON FIELD 2 OF A DESCRIPTOR CARD FOR TASK 'TASKX' IS NOT A VALID CARD TYPE

See Section A.1.3.

A.2.7.4 NO DATA BANK OPTION HAS BEEN SPECIFIED IN THE NAMELIST INPUT

As shown in Table 9.4, one of the four data bank options must be specified in the input NAMELIST.

A.2.7.5 TASK 'TASKY' NAMED IN FIELD 3 OF THE EQUIVALENCE CARD FOR TASK 'TASKX' HAS NOT BEEN DEFINED

See Section A.1.4.

A.2.7.6 THE FIRST DESCRIPTION CARD FOR
TASK 'TASKX' IS NOT A TITLE CARD

See Section A.1.5.

A.2.7.7 OUT OF AVAILABLE SPACE

See Section A.1.7.

A.2.8 Messages Generated by Subroutine TABIN

A.2.8.1 THE INPUT PARAMETERS DO NOT
MATCH THE LAUNCH PARAMETERS
ON THE ATS EPHEMERIS TAPE

INPUT LAUNCH PARAMETERS
DATE 'JAN. 1, 1972'
TIME '15.00' HRS
ATS EPHEMERIS LAUNCH PARAMETERS
DATE 'NOV. 9, 1972'
TIME '15.00' HRS

The launch date and time specified through variables GDATE and TIMEL in the NAMELIST input do not match their counterparts on the ATS Ephemeris Tape.

A.2.8.2 THE TOTAL REQUIRED SPACE IN
THE WA ARRAY IS GREATER THAN
THE DIMENSION OF THE ARRAY

The sum total of locations allocated to all of the tables stored in the WA Array has exceeded the dimension of the array (Section 9.2.1.4). The total number of required locations should be reduced either by reducing the number of locations allocated to some of the tables defined in Array NEWCOM or by removing some of the tables from the ATS Ephemeris Tape (Section 3.2.1).

A.2.8.3 ARRAY LTABLE HAS NO SPACE TO ACCOMMODATE ENTRY 'RESNME'

The names of all Resource Tables stored in the WA Array must be entered in Array LTABLE, (Section 5.1.2). The names are entered into the array in the following sequence:

1. Crewmen CREWA, CREWB, CREWC.
2. The names of all new tables defined in Array NEWCOM (Section 9.2.1.4).
3. The names of all crewmen defined through Array NEWCRW.
4. The names of all Ephemeris Resource Tables stored on the ATS Ephemeris Tape.

The message indicates that the total number of tables defined prior to Table RESNME equals the maximum number of entries permitted in Array LTABLE. To eliminate the problem therefore, the total number of tables being defined must be reduced or the dimension of Array LTABLE must be increased.

A.2.8.4 ARRAY CRWSKL HAS NO SPACE TO ACCOMMODATE 'CREWX'

The names of all crewmen must be entered in Array CRWSKL (Section 7.2). New crewmen (other than CREWA, CREWB, and CREWC) are entered in the CRWSKL Array in the order in which they are specified in input Array NEWCRW. The message indicates that the total number of crewmen defined prior to Crewman CREWX equals the maximum number of entries in the CRWSKL Array. Therefore, the total number of crewmen being defined must be reduced.

A.2.9 Message Generated by Subroutine WINDOW

A.2.9.1 THE TOTAL NUMBER OF RESOURCE REQUIREMENTS AND PERFORMANCE CONSTRAINTS SPECIFIED FOR TASK 'TASKX' IS GREATER THAN THE MAXIMUM NUMBER OF ENTRIES PERMITTED IN THE REQ ARRAY

Each location of Array REQ, an internal working array, is to contain the address of one of the requirement

or constraint sublists in the Task Description List (Figure 5.6). The message indicates that the number of these sublists is greater than the maximum number permitted by the dimension of the array.

The REQ Array is currently dimensioned to 33, the maximum number of requirements that can be specified in a task description (since, as noted in Section 5.3.2.3, the maximum number of Descriptor Cards that can be specified in a task description is 35). Therefore, the only conditions under which this message would be generated would be if the permissible number of Task Description Cards was increased without increasing the dimension of the REQ Array.

A.3 Data Processor Diagnostic Messages

A.3.1 Message Generated in Routine ATSPLT

NO DEPENDENT VARIABLES HAVE
BEEN SPECIFIED

The names of the variables to be plotted have not been specified in the input data (Sections 9.3.1.4 and 9.3.1.5).

A.3.2 Message Generated in Subroutine HPLOT

A.3.2.1

THE DEPENDENT VARIABLE 'NAME'
DOES NOT MATCH ANY OF THE RESOURCE
NAMES LISTED IN ARRAY LTABLE OR
ANY OF THE TASK NAMES LISTED IN
ARRAY ORDER

The alphanumeric combination 'NAME', specified in Array HDEP, cannot be identified. The user should check to insure that the name that appears in Array HDEP identically matches the name of a resource table stored in Array LTABLE or the tame of a task stored in Array ORDER (Section 8.3.1).

A.3.2.2

NO PERFORMANCES OF TASK 'TASKX'
HAVE BEEN SCHEDULED

No performances of Task TASKX were scheduled; hence, no graph of the occurrences of Task TASKX can be generated.

A.3.3 Message Generated in Subroutine VPLOT

A.3.3.1

THE DEPENDENT VARIABLE 'NAME' DOES
NOT MATCH ANY OF THE RESOURCE NAMES
LISTED IN ARRAY LTABLE OR ANY OF
THE TASK NAMES LISTED IN ARRAY ORDER

The alphanumeric combination 'NAME' specified in Array VPOINT or VSHADE cannot be identified. The users should check to insure that the name that appears in the

array input identically matches the name of a resource table stored in Array LTABLE or the name of a task stored in Array ORDER (Section 8.3.1).

A.3.3.2 NO PERFORMANCES OF TASK 'TASKX' HAVE BEEN SCHEDULED

No performances of Task TASKX were scheduled, hence, no graph of the occurrences of Task TASKX can be generated.

A.3.3.3 DEPENDENT VARIABLE 'RESNME' IS LISTED IN LTABLE AS AN ANALOG RESOURCE. ANALOG RESOURCES CANNOT BE DISPLAYED ON PERIODIC PLOTS

The dependent variable RESNME was identified as the name of an analog resource table. Analog resources cannot be displayed on periodic plots (Section 8.4).

DISTRIBUTION

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES:

OFFICIAL FILE COPY
plus one white copy for each additional case referenced

TECHNICAL LIBRARY (4)

NASA Headquarters

H. Cohen/MLQ
J. H. Disher/MLD
W. B. Evans/MLO (2)
J. P. Field, Jr./MLB
T. E. Hanes/MLA
V. N. Huff/MTE (2)
T. A. Keegan/MTE
A. S. Lyman/MR (2)
M. Savage/MLE
W. C. Schneider/ML
D. N. Turner/MF

MSC

B. L. Brady/FS5
R. H. Brown/FM3
J. B. Cotter/CF34
B. E. Ferguson/CF34
J. A. Frere/FS4
J. B. MacLeod/FS
R. E. McAdams/FM3

MSFC

G. B. Hardy/PM-AA-EI
T. E. Telfer/S&E-AERO-MM

Aerospace Corporation

L. T. Stricker
R. R. Wolfe

University of Wisconsin

G. E. Collins

COMPLETE MEMORANDUM TO (Cont'd)

Bellcomm, Inc.

A. P. Boysen, Jr.
J. P. Downs
D. R. Hagner
W. G. Heffron
H. A. Helm
J. J. Hibbert
E. E. Hillyard
N. W. Hinners
R. F. Jessup
D. P. Ling
P. F. Long
K. E. Martersteck
J. Z. Menard
J. M. Nervik
M. P. Odle
S. L. Penn
P. F. Sennewald
R. V. Sperry
W. B. Thompson
J. W. Timko
R. L. Wagner
M. P. Wilson
Division 101 Supervision
Departments 1022, 1024, 1025
Department 1024 Files

BA-146 (8-64)